AMSTRAD

PROGRAMMES EN LANGAGE MACHINE

STEVE WEBB



AMSTRAD

PROGRAMMES EN LANGAGE MACHINE

DANS LA MÊME COLLECTION

Amstrad jeux d'action, P. Monsaut
Amstrad premiers programmes, R. Zaks
Amstrad 56 programmes, S.R. Trost
Amstrad guide du BASIC et de l'AMSDOS, J.-L. Gréco/M. Laurent
Amstrad exploré, J. Braga
Amstrad programmation en assembleur, G. Fagot-Barraly
Amstrad guide du graphisme, J. Winford
Amstrad CP/M 2.2, A. d'Hardancourt
Amstrad astrologie, numérologie, biorythmes, P. Bourgault
Amstrad graphisme en trois dimensions, T. Lachand-Robert
Amstrad Multiplan, Amstrad
Amstrad CP/M plus, A. d'Hardancourt
Amstrad Astrocalc, G. Blanc/P. Destrebecq
Amstrad gagnez aux courses J.-C. Despoine
Amstrad créer de nouvelles instructions, J.-C. Despoine
Amstrad Locoscript, B. Le Dû
Amstrad mise au point des programmes BASIC, C. Vivier/Y. Jacob
Amstrad jeux en assembleur, E. Ravis
Amstrad techniques de programmation des jeux, G. Fagot-Barraly
Amstrad routines en assembleur, J.-C. Despoine
Amstrad jeux de réflexion, G. Fagot-Barraly
Amstrad mieux programmer en assembleur, T. Lachand-Robert
Amstrad gagnez à la Bourse, *J.-C. Despoine/F. Pierre.*
Amstrad PCW 8256/8512 guide du BASIC et de Jetsam, J.-L. Gréco/M. Laurent
Amstrad programmez votre traitement de texte, J.-C. Despoine
Amstrad guide de Logo, A. d'Hardancourt (à paraître)
Amstrad introduction à la programmation en assembleur sous CP/M A. d'Hardancourt (à paraître)
Amstrad Startext, *(logiciel de traitement de texte - à paraître).*

STEVE WEBB

AMSTRAD

PROGRAMMES EN LANGAGE MACHINE



Paris • Berkeley • Düsseldorf

Traduction française de Jean-Louis Gréco

Publié en 1985 en Grande-Bretagne par :

Virgin Books Ltd,
61-63 Portobello Road
LONDON W113DD

Sybex n'est lié à aucun constructeur.

Tous les efforts ont été faits pour fournir dans ce livre une information complète et exacte. Néanmoins, Sybex n'assume de responsabilités ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

ISBN version originale : 0 86369 082 3
ISBN version française : 2-7361-0195-2

Sommaire

Introduction

Ce livre est une introduction à la programmation en langage machine sur l'ordinateur Amstrad CPC 464. Il a été conçu pour permettre un apprentissage progressif, ce qui nécessite une lecture linéaire de l'ouvrage.

La programmation en langage machine n'est pas aussi difficile que ce que l'on en dit parfois. Si l'on possède un minimum de pratique du langage BASIC, les concepts utilisés pour la programmation en langage machine peuvent être assimilés rapidement.

Les premiers chapitres de cet ouvrage traitent de la théorie du langage machine, montrant en particulier comment les nombres sont stockés en mémoire. La description des équivalents fondamentaux de plusieurs instructions BASIC et celle de l'organisation interne de la machine permettent ensuite d'apporter les éléments pour répondre à la question : "Qu'est-ce que le langage machine ?" Le chapitre principal montre comment écrire, dans ce langage, un programme de jeu très simple, en commençant par établir l'organigramme, dont chaque bloc est ensuite converti en de courts sous-programmes en langage machine. Les derniers chapitres décrivent quelques sous-programmes utilitaires et montrent comment tirer parti des possibilités évoluées de votre ordinateur.

1. LE LANGAGE MACHINE

L'unité de taille de la mémoire centrale d'un ordinateur est le kilo octet (ou K). Un kilo octet correspond à un petit peu plus de 1 000 octets (1 024 exactement), de sorte qu'un ordinateur 64 K dispose de 64 000 emplacements mémoire (ou adresses) différents. Chaque emplacement mémoire peut être considéré une boîte ayant un numéro compris entre 0 et 63 999. Ces numéros sont virtuels et servent uniquement à repérer chacune des adresses.

Chaque emplacement mémoire peut contenir un nombre compris entre 0 et 255 inclus. Le fait de ne pouvoir stocker un nombre supérieur à 255 constitue naturellement une limitation, de sorte qu'une méthode a été recherchée pour stocker des nombres plus grands. L'exemple ci-dessous illustre la méthode qui a été retenue.

Supposons que le nombre 29 248 doive être stocké en mémoire centrale de l'ordinateur. La première opération à effectuer consiste à diviser ce nombre par 256 et à retenir la partie entière de cette division. Ainsi, 29 248 divisé par 256 donne 114,25 ; la valeur 114 est donc retenue. Elle est appelée partie de poids fort du nombre à stocker et représente le nombre de multiples entiers de 256 qu'il contient. En multipliant la partie de poids fort du nombre par 256 et en soustrayant le résultat obtenu au nombre à stocker, on obtient le reste de la division entière précédente :

114 X 256 = 29 184
29 248 – 29 184 = 64

64 est appelé partie de poids faible de ce nombre.

Le nombre 29 248 sera stocké en mémoire en chargeant la partie de poids faible (64) à une adresse mémoire choisie et la partie de poids fort (114) à l'adresse suivante. Le stockage d'un nombre supérieur à 255 nécessite donc l'utilisation de deux emplacements mémoire. Ainsi, lorsqu'on lit qu'un nombre supérieur à 255 est stocké par exemple à l'adresse 50000, cela signifie qu'il occupe en fait les adresses 50000 et 50001.

> Quelles sont les parties de poids fort et de poids faible du nombre 45 621 ?
>
> Quel est le nombre qui a 31 pour partie de poids faible et 64 pour partie de poids fort ?

Chaque emplacement mémoire susceptible de contenir un nombre compris entre 0 et 255 correspond à un octet. Un programme ayant une taille de 5 000 octets occupe donc 5 000 emplacements mémoire.

La zone de la mémoire centrale où sont stockés les programmes utilisateur (qu'il s'agisse de programmes écrits en BASIC ou en langage machine) est appelée mémoire RAM (*Random Access Memory/* mémoire à accès sélectif). Le contenu de la mémoire RAM est effacé dès que l'ordinateur n'est plus sous tension. Il existe une autre partie de la mémoire centrale appelée mémoire ROM (*Read Only Memory /* mémoire à lecture seule). Comme son nom l'indique, il n'est possible que de lire le contenu des adresses correspondantes et non de les modifier. Ce contenu n'est pas effacé lorsque l'ordinateur n'est plus sous tension, ce qui est normal car se trouve en particulier à cet emplacement le système d'exploitation sans lequel l'ordinateur ne serait qu'un bloc de métal inerte. Le système d'exploitation contient un certain nombre de sous-programmes en langage machine qui gèrent les fonctions principales de l'ordinateur : ils permettent par exemple de lire le clavier, d'effectuer les calculs fondamentaux et de vérifier la syntaxe des programmes écrits en BASIC.

Le cœur de l'ordinateur est son microprocesseur (de type Z80). Il n'a pas l'intelligence d'un cerveau. La seule chose qu'il puisse faire, ce sont des opérations arithmétiques tout à fait élémentaires. Sa puissance, comparée à celle du cerveau humain, réside dans le fait que le microprocesseur effectue un très grand nombre d'opérations élémentaires par seconde, ce qui peut parfois lui donner l'apparence d'une certaine “intelligence”.

Le microprocesseur ne comprend que les instructions en langage machine. Il existe plus de 600 variantes possibles pour ces instructions. Chaque instruction correspond à un nombre ou à une combinaison de nombres. Par exemple, le nombre 198 correspond à l'instruction commandant d'additionner deux nombres, de sorte que, lorsque le microprocesseur rencontre le nombre 198, il sait que l'opération qu'il doit immédiatement effectuer est une addition (la procédure qu'il utilise pour réaliser cette opération sera décrite plus avant dans l'ouvrage).

Le microprocesseur ne comprenant que les instructions en langage machine, comment peut-il intervenir dans l'exécution de programmes écrits en BASIC ? La réponse à cette question tient à l'existence d'un interpréteur, équivalent informatique de l'inter-

prète dans une langue étrangère. De la même manière qu'un interprète a pour fonction de traduire un discours d'une langue dans une autre, le BASIC s'adresse au microprocesseur par l'intermédiaire d'un interpréteur. On conçoit volontiers que le recours à un interprète ou à un interpréteur a pour inconvénient de ralentir le rythme de la conversation entre deux personnes ou entre le programme BASIC et le microprocesseur.

La possibilité de programmer en langage machine permet, en accédant directement au microprocesseur, de se passer de l'interpréteur, ce qui augmente d'autant la vitesse d'exécution d'un programme. En règle générale, un programme écrit en langage machine s'exécute 50 à 100 fois plus vite que le programme équivalent écrit en BASIC.

2. EQUIVALENCE DE QUELQUES INSTRUCTIONS BASIC EN LANGAGE MACHINE

Certaines commandes BASIC telles que LIST, NEW, RENUM, DELETE n'auraient aucune raison d'être en langage machine et n'ont donc pas d'équivalent. D'autres instructions BASIC telles que RUN, READ, DATA, PRINT, n'ont pas réellement d'équivalent en langage machine, mais nous verrons qu'il est possible de simuler leur action au moyen de certains codes (ou instructions en langage machine).

Les codes présentés ici sont les plus fréquemment utilisés. Leur connaissance suffit à écrire de petits programmes en langage machine.

En BASIC, on est habitué à utiliser des variables telles que A, B, C, ..., X, Y, Z. En langage machine, de telles variables n'existent pas. Leurs équivalents les plus proches sont les registres. Il existe un très petit nombre de registres, et ceux qui sont les plus utilisés sont notés par les lettres :

A, B, C, D, E, H, L

Un autre registre, noté F, sera décrit un peu plus loin. Chaque registre peut être considéré comme représentant un emplacement mémoire ; ainsi, ils ne peuvent contenir que des nombres compris entre 0 et 255. Afin de pouvoir utiliser les registres pour y stocker des nombres plus grands, six d'entre eux ont été groupés par paires, de la manière suivante :

HL
BC
DE

Ces regroupements n'interdisent pas d'utiliser chacun de ces registres individuellement.

L'instruction BASIC :

LET A = 5

a pour code machine équivalent :

LD A, 5

LD est une abréviation du mot LOAD (charger). La signification complète du code ci-dessus est "Charger la valeur 5 dans le registre A".

Chacun des cinq autres registres peut ainsi recevoir une valeur comprise entre 0 et 255 :

LD H, 199 (Charger la valeur 199 dans le registre H)
LD D, 2 (Charger la valeur 2 dans le registre D)

Il a été mentionné précédemment que chaque registre pouvait être considéré comme représentant un emplacement mémoire. Il a été également précisé qu'un nombre tel que 827 avait la valeur 3 (partie entière de la division de 827 par 256) comme partie de poids fort et la valeur 59 comme partie de poids faible. Lorsque le code machine LD HL, 827 est exécuté, la partie de poids fort du nombre 827 est chargée dans le registre H et la partie de poids faible dans le registre L (H et L sont d'ailleurs les initiales des mots anglais *high* (haut) et *low* (bas). De même, dans le registre double BC, B est la partie haute alors que dans la paire DE, E est la partie basse :

BASIC : LET A = B
Signification : Donner à la variable A la même valeur que celle contenue dans la variable B.
Code machine : LD A, B
Signification : Charger dans le registre A la même valeur que celle contenue dans le registre B.

Il est ainsi possible de charger dans un registre simple la même valeur que celle contenue dans n'importe quel autre registre simple. On peut ainsi avoir les codes suivants :

LD A, H
LD E, A
LD H, C

Le code machine LD est probablement le plus utilisé (il est même difficile de concevoir un programme machine sans en faire usage).

BASIC : LET A = A + 5
Signification : Ajouter 5 à la valeur courante de A.
Code machine : ADD A, 5
Signification : Augmenter de 5 le contenu du registre A.

Le registre A est le seul pour lequel il soit possible d'augmenter ainsi directement la valeur qu'il contient. Les codes machine suivants sont par exemple interdits :

ADD B, 9
ADD E, 3

Cette limitation apparente du microprocesseur Z80 peut être facilement contournée, comme décrit plus loin.

BASIC	:	**LET A = A + B**
Signification	:	**Ajouter la valeur de B à celle de A et stocker la valeur obtenue dans A.**
Code machine	:	**ADD A, B**
Signification	:	**Augmenter la valeur courante du registre A de celle contenue dans le registre B.**

Il est ainsi possible d'ajouter à la valeur courante du registre A le contenu de l'un quelconque des sept registres précédemment mentionnés :

ADD A, B
ADD A, C
ADD A, H
ADD A, L
ADD A, D
ADD A, E
ADD A, A

Le dernier code, ADD A, A, a pour effet de doubler la valeur courante du registre A. Toute autre combinaison que celles indiquées ci-dessus est interdite ; c'est par exemple le cas des codes suivants :

ADD B, H
ADD D, C

Les registres doubles peuvent également être additionnés. Cependant, les seules combinaisons permises sont les suivantes :

ADD HL, BC
ADD HL, DE
ADD HL, HL

Le résultat doit donc toujours être stocké dans HL. Le dernier code, ADD HL, HL, a pour effet de doubler la valeur courante du registre HL. Il n'est pas possible d'ajouter directement la valeur contenue dans un registre simple à celle d'un registre double.

Pour ajouter un nombre au contenu d'un autre registre que le registre A, la procédure à suivre est la suivante, en supposant par exemple que l'on veuille ajouter 9 à la valeur courante 14 du registre B :

Étape 1 : **LD A, B**
Le registre A a alors la même valeur que celle contenue dans le registre B.

Étape 2 : **ADD A, 9**
Le contenu du registre A est augmenté de 9.

Étape 3 : **LD B, A**
Le résultat de l'opération est transféré dans le registre B.

Il est ainsi possible, au moyen de trois instructions, d'ajouter un nombre à la valeur contenue dans n'importe lequel des sept registres.

Une autre limitation précédemment mentionnée du microprocesseur Z80 réside dans le fait qu'il est seulement possible d'ajouter au registre A le contenu d'un autre registre. Ainsi, on ne peut écrire :

ADD B, H

Cette restriction peut être contournée de la manière suivante. Supposons que B ait la valeur 5, H la valeur 7 et que l'on veuille additionner le contenu de ces deux registres et stocker le résultat dans B :

Étape 1 : **LD A, B**
Le registre A a ainsi la même valeur que celle du registre B.

Étape 2 : **ADD A, H**
L'addition est effectuée dans le registre A.

Étape 3 : **LD B, A**
Le résultat est transféré dans le registre B.

De nouveau, trois étapes suffisent pour additionner le contenu de deux registres quelconques. Il est aussi possible d'additionner ainsi la valeur d'un registre à lui-même.

De la même manière, on peut additionner le contenu de deux registres doubles quelconques. Pour ajouter par exemple le contenu du registre BC à celui de DE, le résultat devant être stocké dans BC, les trois étapes suivantes doivent être effectuées :

Étape 1 : **LD H, B**
LD L, C
Le registre double HL prend ainsi la valeur du registre BC.

Étape 2 : **ADD HL, DE**
L'addition est effectuée dans HL.

Étape 3 : **LD B, H**
LD C, L
Le résultat est transféré dans le registre double BC.

La plupart des limites du microprocesseur Z80 peuvent être ainsi aisément contournées, un minimum de pratique permettant de sélectionner la méthode la plus efficace pour répondre à un besoin particulier.

BASIC : **LET A = A - 5**
Signification : **Soustraire 5 à la valeur courante de la variable A.**
Code machine : **SUB A, 5**
Signification : **Soustraire 5 à la valeur contenue dans le registre A.**

Comme dans le cas de l'addition, le registre A est le seul auquel on puisse soustraire directement un nombre. Dans le cas des six autres registres, la procédure à employer pour effectuer cette opération est la suivante, en supposant par exemple que la valeur 5 soit à soustraire du registre D :

Étape 1 : **LD A, D**
Le registre A prend ainsi la valeur de D.

Étape 2 : **SUB A, 5**
La soustraction est effectuée dans le registre A.

Étape 3 : **LD D, A**
Le résultat de l'opération est transféré dans le registre D.

BASIC : LET A = A - B
Signification : Soustraire B à A et stocker le résultat dans A.
Code machine : SUB B
Signification : Soustraire la valeur du registre B de celle du registre A.

Là encore, le registre A est le seul auquel il soit ainsi possible de soustraire la valeur contenue dans un autre registre.

Pour soustraire la valeur d'un registre à celle d'un registre autre que A, il est nécessaire de passer par trois étapes semblables à celles décrites ci-dessus.

Il est également possible de soustraire le contenu d'un registre double de celui d'un autre. Cependant, les seules combinaisons permises sont les suivantes :

SBC HL, BC
SBC HL, DE
SBC HL, HL

SBC est une forme particulière de la soustraction. L'instruction SUB ne peut pas être utilisée sur des registres doubles avec le microprocesseur Z80. La signification complète du code SBC sera donnée un peu plus loin. Pour le moment, il suffit de le considérer comme un simple opérateur de soustraction. Une instruction très utilisée, INC, permet d'ajouter (d'INCrémenter) une unité au contenu d'un registre simple ou d'un registre double. Les combinaisons possibles sont les suivantes :

INC A
INC B
INC C
INC D
INC E
INC H
INC L
INC HL
INC BC
INC DE

De manière semblable, l'instruction DEC permet de soustraire (de DECrémenter) une unité au contenu de n'importe lequel des registres (simples ou doubles).

BASIC : **GOTO (numéro de ligne)**
Signification : **Branchement vers la ligne spécifiée.**
Code machine : **JP (adresse mémoire)**
Signification : **Saut à l'adresse mémoire spécifiée.**

Le code machine JP a une fonction très proche de l'instruction BASIC "GOTO", l'adresse mémoire remplaçant ici le numéro de ligne. Le chapitre suivant montrera comment les codes machine sont stockés en mémoire. L'utilisation de ceux-ci deviendra alors plus claire.

BASIC : **GOSUB (numéro de ligne)**
Signification : **Branchement vers un sous-programme dont le numéro de la première ligne est spécifié. Le sous-programme doit se terminer par une instruction RETURN.**
Code machine : **CALL (adresse mémoire)**
Signification : **Branchement vers un sous-programme commençant à l'adresse spécifiée. Comme les sous-programmes en BASIC, ceux écrits en langage machine doivent se terminer par une instruction RET.**
BASIC : **IF A = 5 THEN (GOTO/GOSUB/LET/etc.)**
Signification : **Si A = 5, alors exécuter l'instruction ou le branchement spécifié par la clause THEN.**

En langage machine, l'équivalent des instructions IF... THEN existe, mais celles-ci ne sont pas implantées de la même manière qu'en BASIC. La condition IF se rapporte toujours, en langage machine, au résultat de la dernière opération qui a été effectuée. L'équivalent typique d'une instruction IF... THEN est :

CALL Z, (adresse mémoire spécifiée)

Cette instruction signifie que, si le résultat du dernier calcul est zéro, le sous-programme se trouvant à l'adresse spécifiée doit être appelé.

La liste des codes machine équivalant aux instructions IF... THEN les plus fréquemment utilisées est donnée ci-dessous :

CALL NZ, nn

Si le résultat du dernier calcul effectué est différent de zéro, le sous-programme se trouvant à l'adresse nn est appelé.

CALL M, nn

Si le résultat du dernier calcul effectué est négatif, le sous-programme se trouvant à l'adresse nn est appelé.

CALL P, nn

Si le résultat du dernier calcul effectué est positif, le sous-programme se trouvant à l'adresse nn est appelé.

JP Z, nn

Si le résultat du dernier calcul effectué est zéro, l'exécution du programme est déroutée vers l'adresse nn.

JP NZ, nn

Si le résultat du dernier calcul effectué est différent de zéro, l'exécution du programme est déroutée vers l'adresse nn.

JP M, nn

Si le résultat du dernier calcul effectué est négatif, l'exécution du programme est déroutée vers l'adresse nn.

JP P, nn

Si le résultat du dernier calcul effectué est positif, l'exécution du programme est déroutée vers l'adresse nn.

BASIC	:	**FOR A = 1 TO 100**
		Instruction(s) devant être exécutée(s) 100 fois.
		NEXT A
Signification	:	**Effectuer 100 fois la boucle.**
Code machine	:	**LD A, 100**
		Instruction(s) devant être exécutée(s) 100 fois.
		SUB A, 1
		JP NZ, adresse de la première instruction de la boucle.

Ainsi, pour simuler la boucle FOR/NEXT indiquée, la valeur 100 doit tout d'abord être chargée dans le registre A. Cette valeur représente le nombre de fois où les instructions se trouvant à l'intérieur de la boucle doivent être exécutées. Après chaque exécution de la série d'instructions, la valeur 1 est soustraite du contenu du registre A. Si le résultat de cette soustraction est NZ (Non Zéro), l'exécution est redirigée vers la première instruction de la

boucle. Le processus se poursuit jusqu'à ce que la soustraction d'une unité au contenu du registre A donne pour résultat zéro,ce qui signifie alors que la boucle a été exécutée autant de fois qu'il était spécifié.

Les instructions BASIC "PEEK" et "POKE" sont, dans ce langage évolué, celles qui se rapprochent le plus des codes machine. Elles permettent en effet un accès direct au contenu de la mémoire.

BASIC	**: LET A = PEEK (40000)**
Signification	**: Assigner à la variable A la valeur contenue a l'adresse mémoire 40000.**
Code machine	**: LD A, (40000)**
Signification	**: Charger la valeur se trouvant à l'adresse 40000 dans le registre A.**

Si la valeur 81 est stockée à l'adresse 40000 et que le code LD A, (40000) est exécuté, la valeur 81 sera chargée dans le registre A. Ce registre est le seul registre auquel il soit possible d'assigner ainsi directement une valeur se trouvant à une adresse mémoire. Par exemple, l'instruction suivante est incorrecte :

LD D, (40000)

Par contre, les trois registres doubles peuvent être utilisés de cette manière pour simuler une instruction PEEK.

Code machine : LD HL, (40000)

Cette instruction a pour effet de charger dans le registre L la valeur se trouvant à l'adresse 40000 et dans le registre H la valeur se trouvant à l'adresse 40001. Cela découle naturellement de la manière dont les nombres supérieurs à 255 sont stockés en mémoire centrale de l'ordinateur.

> Si les valeurs 5 et 15 sont respectivement stockées aux adresses 40000 et 40001, quelle sera la valeur contenue dans le registre HL après que le code machine LD HL,(40000) aura été exécuté ?

BASIC : **POKE(40000),A**
Signification : **Charger la valeur de la variable A à l'adresse 40000.**
Code machine : **LD (40000),A**
Signification : **Charger la valeur se trouvant dans le registre A à l'adresse 40000.**

Le registre A est le seul registre simple dont la valeur puisse être chargée directement à un emplacement mémoire quelconque. Par contre, les trois registres doubles peuvent être ainsi utilisés pour simuler l'action d'une instruction POKE

Code machine : **LD (40000),HL**
Signification : **Charger à l'adresse 40000 la valeur contenue dans le registre L et à l'adresse 40001 la valeur contenue dans le registre H.**

> Si le registre HL contient la valeur 35621, quel sera le contenu des adresses mémoire 40000 et 40001 après que le code machine LD (40000), HL aura été exécuté ?

BASIC : **LET A = 5**
LET B = 40000
POKE(B), A
Signification : **Charger la valeur de la variable A à l'adresse mémoire spécifiée par B.**
Codes machine : **LD A, 5**
LD HL, 40000
LD (HL), A
Signification : **Charger la valeur contenue dans le registre A à l'adresse mémoire correspondant à la valeur contenue dans HL.**

BASIC : **LET B = 40000**
LET A = PEEK(B)
Signification : **Assigner à la variable A la valeur se trouvant à l'adresse spécifiée par B.**
Codes machine : **LD HL, 40000**
LD A, (HL)
Signification : **Charger dans le registre A la valeur se trouvant à l'adresse correspondant à la valeur contenue dans HL.**

Le registre A est le seul registre simple qui puisse être ainsi utilisé avec une adresse spécifiée par le contenu de l'un des trois registres doubles. Les combinaisons permises sont donc les suivantes :

LD A, (HL)
LD A, (BC)
LD A, (DE)

Les six autres registres simples ne peuvent être ainsi chargés que lorsque l'adresse est spécifiée par le registre double HL, au moyen des combinaisons suivantes :

LD B, (HL)
LD C, (HL)
LD D, (HL)
LD E, (HL)
LD H, (HL)
LD L, (HL)

Codes machine : PUSH
POP

Ces deux instructions n'ont pas vraiment d'équivalent en BASIC. Elles sont par contre essentielles pour la programmation en langage machine et l'exemple suivant va permettre d'illustrer leur fonction. Supposons qu'un programme utilise les sept registres de la manière suivante :

HL
BC
DE
A

Si l'on voulait exécuter plusieurs fois cette suite d'instructions en BASIC, on utiliserait une boucle FOR/NEXT. Comme indiqué précédemment, une telle boucle peut être simulée par des instructions en langage machine. Dans l'exemple présent, ces instructions sont les suivantes :

```
                          LD A,8 (nombre de fois où la boucle doit être répétée)
(début du sous-programme) HL
                          BC
                          DE
                          A
                          SUB A,1
                          JP NZ, (début du sous-programme)
                          END
```

En l'état, un tel programme ne tournerait pas. En effet, cela revient à définir une boucle FOR/NEXT telle que FOR A = 1 TO 8 et à utiliser la variable A non seulement comme compteur, mais aussi comme paramètre à l'intérieur de la boucle. Dans ces conditions, la valeur de A contrôlant la boucle serait modifiée par son utilisation à l'intérieur de celle-ci. En BASIC, un tel problème est facilement évité car un très grand nombre de variables différentes sont disponibles. Il n'en est pas de même en langage machine et les instructions PUSH et POP sont destinées à surmonter cette difficulté.

Le sous-programme présenté ci-dessous montre comment ces instructions doivent être utilisées pour rendre opérant le sous-programme précédent :

```
                          LD A, 8
(début du sous-programme) PUSH A
                          HL
                          BC
                          DE
                          A
                          POP A
                          SUB A, 1
                          JP NZ, (début du sous-programme)
```

Après que la valeur 8 a été chargée dans le registre A, l'instruction PUSH A est exécutée. Cela signifie que la valeur courante de ce registre est mise de côté, dans un endroit appelé "pile". Les instructions suivantes peuvent alors être exécutées sans risque de perdre cette valeur particulière. Une fois que le sous-programme a été exécuté, l'instruction POP A permet de réaffecter au registre A sa valeur originale. PUSH est appelée instruction d'"empilage" et POP instruction de "dépilage".

On peut ainsi empiler ou dépiler n'importe lequel des registres doubles. Par contre, ces opérations ne peuvent être effectuées sur les registres simples, contrairement à ce que peut laisser croire l'exemple donné ci-dessus. Celui-ci était uniquement destiné à éviter toute confusion, mais le code machine PUSH A n'existe pas. Par contre, il est possible d'empiler un registre double nommé AF. Le registre F est un registre particulier qui ne peut être utilisé de la même manière que les sept autres registres simples. Il est cependant possible d'écrire des programmes très élaborés en langage machine sans avoir recours à ce registre F.

La fonction première du registre F est de servir au microprocesseur Z80 à indiquer le résultat de différents calculs. En fonction de la valeur contenue dans ce registre, le microprocesseur peut par exemple indiquer si le résultat du dernier calcul effectué était nul, positif ou négatif, ou s'il comportait une retenue. C'est précisément cette retenue (ou report) qui fait la différence entre les codes machine SUB (Soustraction) et SBC (Soustraction avec retenue). Si l'indicateur de report est positionné, la retenue est prise en compte dans la soustraction effectuée par l'instruction SBC. L'utilisation du registre F est assez délicate pour les programmeurs peu familiers du langage machine. Aussi ne sera-t-elle pas davantage évoquée dans la suite de cet ouvrage.

Code machine : EX

Ce code permet d'échanger les valeurs des registres spécifiés :

EX DE,HL

signifie échanger les valeurs des registres DE et HL. Ce code peut être utilisé par exemple lorsqu'on désire ajouter la valeur de BC à DE et faire apparaître le résultat dans DE :

```
EX DE,HL
ADD HL,BC
EX DE,HL
```

Il y a quatre autres codes EX. Cependant, ces codes ne seront pas d'une grande utilité pour les débutants.

Code machine : CPr

Ce code permet de comparer le registre A avec les autres registres simples. Si le registre A a la même valeur qu'un autre registre, on peut appeler ou effectuer un branchement à un sous-programme :

CP A,E
CALL Z,sous-programme
Suite du programme

Si les registres A et E ont la même valeur, le sous-programme est appelé.

Le registre A peut aussi être comparé à un nombre :

CP A,126
CALL Z,sous-programme
Suite du programme

Si le registre A contient la valeur 126, le sous-programme est appelé.

3. STOCKAGE DES CODES MACHINE EN MÉMOIRE

Jusqu'à présent, seule la manière dont les nombres sont stockés en mémoire a été décrite. Il a été cependant mentionné que les codes machine étaient représentés par un nombre unique ou par une combinaison de nombres. En pratique, cela signifie que certains codes machine sont représentés par un seul nombre compris entre 0 et 255 et que d'autres le sont par deux nombres compris entre les mêmes limites.

Les nombres représentant les codes machine sont stockés en mémoire de manière tout à fait identique à celle utilisée pour le stockage de n'importe quel autre nombre. Certains codes machine nécessitent un seul emplacement mémoire (c'est-à-dire un seul octet), d'autres en nécessitent deux.

Si la première instruction d'un programme était INC A (incrémenter le registre A), la première chose à connaître serait la valeur du numéro correspondant à cette instruction. Il s'agit en fait de 60. On aurait ensuite besoin de savoir à quelle adresse mémoire doit commencer ce programme. Supposons que ce soit l'adresse 40000. La valeur 60 doit donc être chargée à l'adresse 40000. (La procédure à suivre pour réaliser cette opération sera expliquée un peu plus loin.) Une fois le premier code machine chargé en mémoire centrale, le numéro correspondant à la seconde instruction, soit par exemple 52 pour le code INC HL, doit être chargé à l'adresse suivante, soit ici 40001.

La totalité du programme est ainsi construite pas à pas en stockant aux adresses consécutives les numéros correspondant aux instructions successives. Les deux codes machine donnés en exemple s'étendent sur un seul octet, c'est-à-dire qu'ils n'occupent chacun qu'un seul emplacement mémoire. Cependant, d'autres instructions telles que ADD A,5 s'étendent sur deux octets. Le premier emplacement mémoire contient le numéro correspondant à l'instruction ADD A, tandis que l'emplacement mémoire suivant contient la valeur qui doit être ajoutée au contenu du registre A. Si ADD A,5 était la première instruction d'un programme commençant à l'adresse 40000, la valeur 198 (qui est le numéro correspondant au code machine ADD A) devrait être chargée à l'adresse 40000, la valeur 5 étant chargée à l'adresse 40001.

Lorsque le programme est lancé, le microprocesseur examine le contenu de la première adresse mémoire, en l'occurrence ici ADD A. Il regarde ensuite, à l'adresse 40001, quelle est la valeur qui doit être ajoutée à celle du registre A. Une fois l'addition effec-

tuée, le microprocesseur examine le code machine de l'instruction se trouvant à l'adresse suivante, soit ici 40002.

Certains codes machine s'étendent eux-mêmes sur deux octets et doivent être suivis d'un ou deux arguments occupant chacun un octet. Ces instructions en langage machine s'étendent donc sur trois ou quatre octets au total. Là encore, ces octets doivent être stockés à des adresses consécutives en mémoire centrale.

Une différence essentielle par rapport au BASIC réside dans le fait que ces instructions du langage machine ne sont pas précédées d'un numéro de ligne. Elles sont stockées à des adresses consécutives en mémoire centrale. Le microprocesseur garde en permanence un pointeur sur l'adresse de l'instruction qu'il est en train d'exécuter. Le pointeur se déplace à l'adresse de l'instruction suivante une fois que la première a été exécutée. En dépit de l'absence de numéros de lignes, il est néanmoins possible d'avoir l'équivalent des instructions BASIC de déroutement GOTO et GOSUB. Au lieu d'être dirigée vers un numéro de ligne, l'exécution du programme est déroutée vers une adresse mémoire.

Lorsque l'on écrit un programme en langage machine, les nombres ne sont pas tapés en utilisant le système de notation décimale classique. A la place est utilisée une notation dite "hexadécimale" (HEX en abrégé), c'est-à-dire une notation se rapportant au système à base 16. Les quelques exemples ci-dessous illustrent la correspondance entre les systèmes décimal et hexadécimal :

Décimal	**Hexadécimal**
0	**00**
9	**09**
10	**0A**
15	**0F**
16	**10**
255	**FF**

Un tableau complet de conversion est donné en annexe. Dans cet ouvrage, afin d'éviter toute confusion, les nombres exprimés en notation décimale sont suivis de la lettre d et ceux exprimés en notation hexadécimale sont suivis de la lettre h :

8d = 8 décimal
12h = 12 hexadécimal

Quel est l'équivalent décimal de E 3h ?

Si FBh est la partie de poids fort d'un nombre et CBh sa partie de poids faible, quel est ce nombre exprimé en décimal ?

Le tableau de conversion décimal/hexadécimal donné à la fin de cet ouvrage peut être utilisé pour répondre à ces questions.

La conversion d'un nombre d'un système en un autre devient très facile lorsque l'on possède un minimum de pratique. Un des avantages du système hexadécimal réside dans sa plus grande facilité d'utilisation en informatique. Deux chiffres hexadécimaux suffisent pour représenter n'importe quel nombre décimal compris entre 0 et 255 (soit des nombres exprimés par un, deux ou trois chiffres décimaux).

Il existe plusieurs méthodes permettant d'entrer des nombres ou des codes machine en mémoire centrale de l'ordinateur. Le programme BASIC présenté ci-dessous peut être utilisé pour entrer et vérifier très rapidement un programme écrit en langage machine. Ce programme BASIC doit être soigneusement tapé et, après avoir vérifié qu'il ne contient pas d'erreur de recopie, être sauvegardé sur cassette au moyen de l'instruction :

SAVE " ENTHEX"

```
10 MODE 1
20 MEMORY 29999
30 CLS
40 LOCATE 1,1
50 PRINT"TAPER CAPS LOCK"
60 LOCATE 1,4
70 PRINT"TAPER E POUR ENTRER 1 CODE HEXA
DECIMAL"
80 LOCATE 1,8
90 PRINT"TAPER C POUR VERIFIER LES CODES
"
100 LOCATE 1,12
110 PRINT"TAPER X POUR VERIFIER LE TOTAL
"
120 LOCATE 1,16
130 PRINT"TAPER Q POUR QUITTER"
140 INPUT A$
150 IF A$="Q" THEN STOP
160 IF A$="E" THEN GOTO 370
170 IF A$="C" THEN GOTO 770
180 IF A$="X" THEN GOTO 200
190 GOTO 140
200 CLS
210 LOCATE 1,1
220 PRINT"ADRESSE DE DEPART"
230 INPUT SA
240 LOCATE 1,5
250 PRINT"ADRESSE FINALE"
260 INPUT EA
270 LET D=0
280 FOR C=SA TO EA
290 LET D=D+PEEK(C)
300 NEXT C
310 CLS
320 PRINT"TOTAL =";D
330 LOCATE 1,20
340 PRINT"TAPER M POUR RETOURNER AU MENU
"
```

```
350 IF INKEY$<>"M" THEN GOTO 350
360 GOTO 30
370 CLS
380 LOCATE 1,1
390 PRINT"TAPER CAPS LOCK"
400 LOCATE 1,4
410 PRINT"ADRESSE DE DEPART"
420 INPUT S
430 IF S<30000 THEN GOTO 750
440 LET A$=""
450 LOCATE 1,25
460 LET ET=S
470 IF A$="" THEN INPUT A$
480 LET BAD=0
490 IF A$="M" THEN GOTO 30
500 LET E=LEN(A$)
510 LET E=E-1
520 LET C$=A$
530 FOR D=1 TO E STEP 2
540 LET B$=LEFT$(C$,2)
550 LET C=VAL("&H"+B$)
560 IF C=0 THEN GOSUB 730
570 LET C$=MID$(C$,3)
580 NEXT D
590 IF BAD=1 THEN GOTO 690
600 LOCATE 1,25:PRINT S;" ":A$
610 LET B$=LEFT$(A$,2)
620 IF LEN(B$)=1 THEN GOTO 690
630 LET C=VAL("&H"+B$)
640 POKE(S),C
650 LET S=S+1
660 LET A$=MID$(A$,3)
670 IF A$="" THEN GOTO 460
680 GOTO 610
690 LOCATE 1,25
700 LET S=ET
710 LET A$=""
720 GOTO 460
```

```
730 IF B$<>"00" THEN LET BAD=1
740 RETURN
750 PRINT"ADRESSE DE DEPART >= A 30000"
760 GOTO 400
770 LET ND=0
780 CLS
790 LOCATE 1,1
800 PRINT"ADRESSE DE DEPART"
810 INPUT SA
820 LOCATE 1,5
830 PRINT"ADRESSE FINALE"
840 INPUT EA
850 CLS
860 IF SA+20>EA THEN GOTO 960
870 FOR C=SA TO SA+20
880 GOSUB 1000
890 NEXT C
900 IF ND=1 THEN 1050
910 IF C>EA THEN GOTO 1050
920 PRINT"TAPER M POUR CONTINUER"
930 LET SA=C
940 IF INKEY$<>"M" THEN GOTO 940
950 GOTO 860
960 FOR C=SA TO EA
970 GOSUB 1000
980 NEXT C
990 GOTO 900
1000 IF PEEK(C)<16 THEN GOTO 1030
1010 PRINT C;" ";HEX$(PEEK(C))
1020 RETURN
1030 PRINT C;" 0";HEX$(PEEK(C))
1040 RETURN
1050 PRINT"TAPER M POUR RETOURNER AU MEN
U"
1060 IF INKEY$<>"M" THEN GOTO 1060
1070 GOTO 30
```

UTILISATION DU PROGRAMME ENTHEX

Pour charger le programme ENTHEX, il suffit de taper LOAD "". Lorsque le programme est chargé, on tape RUN et le menu suivant apparaît :

Tapez la touche E pour entrer un code hexadécimal
Tapez la touche C pour vérifier un code
Tapez la touche X pour vérifier tous les codes
Tapez la touche Q pour arrêter

Le programme suivant va nous permettre de tester le programme BASIC :

Codes hexadécimaux	Équivalent assembleur
3E11	LD A,17
C605	ADD A,5
326275	LD (30050) ,A
C9	RET

La première ligne signifie : charger le registre A avec la valeur 17. La deuxième : ajouter 5 au registre A. La troisième : stocker la nouvelle valeur de A à l'adresse 30050. La quatrième : retourner au BASIC.

1. Taper la touche E.
2. Appuyer sur la touche de verrouillage des majuscules.
3. L'adresse de départ à entrer est 30000.
4. Les codes hexadécimaux peuvent être maintenant entrés. Le premier est 3E11 (presser ensuite la touche RETURN). Cela correspond aux deux octets se trouvant aux adresses 30000 et 30001.
5. Entrer de la même manière les trois autres lignes en tapant après chacune d'elles la touche RETURN.
6. Après que la dernière ligne a été entrée et la touche RETURN activée, il suffit de taper M suivi de la touche RETURN pour retourner au menu.

Il faut ensuite vérifier que les codes ont été correctement entrés. Pour ce faire, on tape la touche C suivie de RETURN. Le programme demande alors l'adresse de départ et l'adresse finale (30000 et 30007). L'écran affiche alors les adresses mémoire suivies de leur contenu. Il faut soigneusement vérifier les valeurs afin qu'elles correspondent à celles citées précédemment. Si une ou plusieurs valeurs sont fausses, il faut recommencer la procédure. On retourne ensuite au menu en tapant la touche M. La dernière vérification consiste à contrôler la somme de tous les octets. On tape la touche X suivie de RETURN. Le programme demande alors les adresses de départ et de fin (30000 et 30007). L'écran affiche :

TOTAL = 748

Si ce n'est pas le cas, c'est qu'un code est incorrect ou que le programme ENTHEX (hypothèse moins probable) a été mal tapé.

Nous allons maintenant tester ce petit programme. On tape la touche Q pour arrêter le programme ENTHEX, puis :

```
1000 CALL 30000
1005 PRINT PEEK (30050)
```

On tape maintenant en mode direct GOTO 1000. Le nombre 22 doit s'afficher sur l'écran (résultat de l'addition de 17 et de 5).

La procédure à utiliser pour entrer et vérifier tous les programmes en langage machine se trouvant dans cet ouvrage est identique à celle qui vient d'être détaillée. Elle ne sera donc pas expliquée chaque fois. Il est important de noter les adresses initiale et finale de ces programmes ainsi que la somme des codes hexadécimaux mis en jeu. De même les tests à effectuer sont toujours identiques à ceux qui viennent d'être décrits. D'autre part, quelques lignes de programme BASIC peuvent également être indiquées en tête de chaque listing ; elles commencent toujours à la ligne 1000 et sont exécutables au moyen de la commande GOTO 1000.

Pour sauvegarder un programme écrit en code machine, on tape la commande :

SAVE"NOM",B,Adresse de départ,Longueur

Pour sauvegarder le petit programme de test, il suffit de taper :

SAVE"Add",B,3000,8

Pour le recharger, il suffit de taper LOAD"".

Nous allons maintenant étudier quelques routines qui seront utiles dans l'élaboration de programmes personnels. La ROM de l'Amstrad dispose d'environ 200 routines accessibles par les programmeurs. Nous allons en décrire quelques-unes des plus usuelles.

INITIALISATION DU MODE ÉCRAN

L'Amstrad possède trois modes écran (0, 1, 2). Chacun de ces trois modes peut être facilement mis en œuvre en appelant une routine qui se trouve à l'adresse 48142. Avant l'appel, le registre A doit contenir la valeur correspondant au mode écran désiré (0, 1, 2). La routine suivante décrit la mise en œuvre du mode 2 :

Adresse de départ **30000**
Adresse finale **30005**
Total **672**

```
3E02      LD     A,2
CD0EBC    CALL   48142
C9        RET
```

La routine peut être testée en tapant :

```
1000 CALL 30000
1005 STOP
```

POSITION DU CURSEUR

Pour afficher un caractère à l'écran, la première chose à faire est de positionner le curseur à l'endroit désiré. Il est possible d'initialiser la position du curseur à l'aide d'une routine se trouvant à l'adresse 47989. Avant l'appel de la routine, le registre H doit contenir la coordonnée verticale et le registre L la coordonnée horizon-

tale. La routine ci-après initialise le mode écran 2 puis place le curseur en colonne 5 rangée 10 :

Adresse de départ	**30000**
Adresse finale	**30012**
Total	**1280**

```
3E02     LD      A,2
CD0EBC   CALL    48142
2605     LD      H,5
2E0A     LD      L,10
CD75BB   CALL    47989
C9       RET
```

La routine peut être testée en tapant :

```
1000 CALL 30000
1005 GOTO 1005
```

AFFICHAGE D'UN CARACTÈRE

Après avoir positionné le curseur, il faut afficher le caractère. La routine se trouvant à l'adresse 47962 effectue cette opération. Avant d'appeler la routine, le registre A doit contenir un nombre entre 0 et 255 (représentant le code du caractère à afficher). La lettre A a pour code 65. La rouine suivante initialise l'écran en mode 2, positionne le curseur en colonne 5 rangée 10, puis affiche la lettre A :

Adresse de départ	**30000**
Adresse finale	**30017**
Total	**1889**

```
3E02     LD      A,2
CD0EBC   CALL    48142
2605     LD      H,5
2E0A     LD      L,10
CD75BB   CALL    47989
3E41     LD      A,65
CD5ABB   CALL    47962
C9       RET
```

La routine peut être testée en tapant :

```
1000 CALL 30000
1005 GOTO 1005
```

LECTURE DU CLAVIER

La plupart des programmes utilisent des entrées à partir du clavier. La routine permettant la lecture du clavier se trouve à l'adresse 47902. Cette routine n'indique pas immédiatement quelle touche vient d'être tapée, il faut tout d'abord savoir si une touche spécifique vient d'être tapée. Le registre A doit contenir le code de la touche qui doit être testée ; la routine est ensuite appelée. La routine suivant le test doit appeler un sous-programme si le résultat du test est différent de 0. Ce sous-programme est bien sûr celui qui doit être exécuté si la touche a bien été tapée.

Dans l'exemple qui suit, l'instruction après la routine de test effectue un saut si le résultat du test est égal à 0 au début du programme. Cela permet au programme d'effectuer une boucle jusqu'à ce que la touche spécifiée soit tapée.

Adresse de départ **30000**
Adresse finale **30008**
Total **1121**

```
3E45      LD      A,65
CD1EBB    CALL    47902
CA3075    JP      Z,START
C9        RET
```

La routine est testée en tapant :

```
1000 CALL 30000
1005 PRINT "HOOO"
```

BOUCLES

Les programmes en langage machine sont très rapides, il est parfois nécessaire d'y incorporer des boucles de temporisation. La routine suivante montre une boucle de temporisation simple.

Le registre A contient une valeur correspondant à la durée de la boucle. L'instruction suivante permet de décrémenter le registre A, et fait office de compteur. Ensuite on vérifie que la valeur contenue dans le registre A n'est pas égale à 0 ; si c'est le cas, le programme se poursuit. Cette boucle se réitère jusqu'à ce que la valeur contenue dans A soit égale à 0 :

```
     LD A,35
LOOP DEC A
     JP NZ,LOOP
```

Même avec une valeur égale à 255, la boucle de temporisation s'exécute très rapidement. On peut cependant augmenter la longueur de cette boucle d'attente en utilisant le registre double BC. Le registre BC peut contenir une valeur comprise entre 1 et 65535 :

```
      LD BC, 1743
DELAY DEC C
      JP NZ,DELAY
      DEC B
      JP NZ,DELAY
```

La plupart des programmes utilisent plusieurs boucles de temporisation à différents endroits du programme. Afin de mieux gérer l'espace mémoire disponible, il est utile d'écrire une routine permettant des temporisations variables :

```
DELAY DEC C
      JP NZ,DELAY
      DEC B
      JP NZ,DELAY
      RET
```

On remarque que cette routine est pratiquement identique à la précédente, mais que la valeur de BC n'a pas été déclarée au début et qu'une instruction RET est apparue à la fin du programme. Cette méthode sera utilisée dans l'écriture du programme “Les envahisseurs de l'espace”.

4. ÉCRITURE D'UN PROGRAMME EN LANGAGE MACHINE : LES ENVAHISSEURS DE L'ESPACE

Le programme de jeu proposé ici est simple. Cependant, tous les principes mis en œuvre sont applicables à la réalisation de programmes plus sophistiqués. Un programme peut être décomposé en une succession de blocs, chaque bloc étant ensuite lui-même découpé en une suite de sous-programmes. Selon ce principe, le lecteur doit être capable, à la fin de ce chapitre, d'écrire ses propres programmes en langage machine.

Ce chapitre commence par une introduction à la notion d'organigramme, déjà familière aux lecteurs ayant pratiqué le langage BASIC. Il se poursuit par la description de l'organisation d'une carte mémoire. Cette étape, qui n'est pas nécessaire en BASIC, est indispensable lorsque l'on commence à écrire ses propres programmes en langage machine. Chacun des blocs de l'organigramme sera ensuite détaillé. L'un de ces blocs concerne par exemple le déplacement d'une balle à l'écran. La méthode mise en œuvre pour réaliser ce déplacement sera expliquée et l'organigramme d'un sous-programme interne sera également présenté à cette occasion.

Tous les sous-programmes ont été écrits afin de pouvoir être testés individuellement, ou en liaison avec d'autres sous-programmes qui l'ont déjà été. Le programme complet peut ainsi être examiné étape par étape et l'on peut voir quelle est exactement la fonction de chacun de ses sous-programmes constitutifs.

ORGANIGRAMME

L'organigramme est une description graphique des différentes opérations devant être effectuées par le programme, le schéma devant en principe montrer l'articulation logique entre les différents blocs. L'organigramme présenté ci-contre est dépouillé de toute symbolisation pour ne laisser apparaître que le découpage en blocs du programme "Les envahisseurs de l'espace".

INITIALISATION

30 Octets

Boucle principale
Appel des sous-programmes
Déplacement de l'envahisseur
Déplacement du fusil
Déplacement de la balle
Branchement au début de la boucle

55 Octets

Initialisation
Affichage envahisseur
Affichage fusil
Mode écran 0
Retour

100 Octets

Déplacement envahisseur
Effacement envahisseur
Une balle a touché ?
Si oui branchement à la routine "Touché"
Déplacement envahisseur dans direction appropriée

Touché
Affichage envahisseur mort
Retour

70 Octets

Déplacement de la balle
Une balle a été tirée ?
Sinon, branchement à la routine de test.
Effacement de la balle
La balle a atteint le sommet de l'écran ?
Si oui retour
Déplacement de la balle d'une position
Retour

Test de tir
Le bouton de tir a été pressé ?
Sinon, temporisation puis retour à la boucle principale
Affichage de la balle au bout du fusil
Retour

70 Octets

Déplacement du fusil
La touche gauche est tapée ?
Sinon, branchement à la routine pour vérifier l'état de la touche droite
Déplacement du fusil d'une position vers la gauche
Retour

La touche droite est tapée ?
Sinon, retour à la boucle principale
Déplacement du fusil d'une position vers la droite
Retour

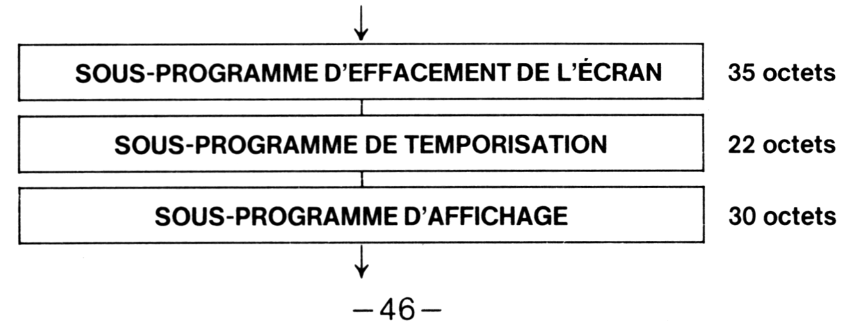
8 Octets

Temporisation
Retour

LA CARTE MÉMOIRE

Il a déjà été précisé qu'il n'existait pas de numéros de lignes dans un programme écrit en langage machine. Cela signifie que si l'on souhaite ajouter une instruction dans un programme déjà écrit, il est nécessaire de déplacer toutes les instructions se trouvant après celle que l'on veut insérer, ce qui nécessite de modifier les adresses absolues données comme arguments aux instructions telles que CALL et JUMP. Supposons par exemple qu'un sous-programe de temporisation DELAI ait été écrit à partir de l'adresse 40000. Pour accéder à ce sous-programme, le programme principal contient une instruction CALL 40000. Si, pour une raison ou pour une autre, une simple instruction de trois octets doit être ajoutée à la fin du sous-programme précédent se terminant normalement à l'adresse 39999, les adresses 40000, 40001 et 40002 devront être utilisées. Il est donc nécessaire de déplacer le sous-programme DELAI afin qu'il commence à l'adresse 40003. Cela fait, les instructions CALL 40000 se trouvant dans le programme devront être remplacées par CALL 40003. L'utilisation d'une telle méthode pour écrire un programme en langage machine serait extrêmement laborieuse et prendrait un temps considérable.

Ces difficultés peuvent être évitées en créant une carte des emplacements mémoire à partir de l'organigramme du programme. Cela nécessite de connaître le nombre approximatif d'octets occupés par chacun des sous-programmes. Les emplacements mémoire peuvent alors être alloués aux sous-programmes en laissant quelques octets libres entre la fin d'un sous-programme et le début du suivant. Ces quelques octets supplémentaires permettront d'étendre, si nécessaire, un sous-programme sans qu'il y ait besoin pour autant de modifier tout le programme. L'organigramme présenté ci-dessous montre comment créer une carte mémoire (dans le cas d'un programme très simple) :



Dans cet organigramme, la taille de chaque sous-programme est écrite en regard de ceux-ci. Le sous-programme d'effacement de l'écran peut ainsi être chargé à partir de l'adresse 40000, le sous-programme de temporisation à partir de l'adresse 40050 et le sous-programme d'affichage à partir de l'adresse 40080. Quelques octets sont laissés libres entre deux sous-programmes consécutifs, facilitant ainsi leur extension éventuelle.

Dans cet esprit, la taille approximative de chacun des sous-programmes constitutifs du programme "Les envahisseurs de l'espace" est indiquée sur l'organigramme correspondant. A partir de ces informations, il est possible de construire la carte mémoire présentée ci-dessous. La colonne de gauche contient le nom de chaque sous-programme, celle de droite l'adresse de départ correspondante :

BOUCLE PRINCIPALE **30000**
INITIALISATION **30041**
DÉPLACEMENT ENVAHISSEUR **30107**
DÉPLACEMENT DE LA BALLE **30227**
DÉPLACEMENT DU FUSIL **30311**
BOUCLE DE TEMPORISATION **30395**
VARIABLES **31000**

Une zone mémoire a été affectée aux variables. La fonction de celles-ci sera brièvement expliquée.

DÉFINITIONS

L'explication des quelques termes ci-dessous facilitera la compréhension du programme.

1. **GAMELOOP** (boucle de jeu) : Il s'agit de la boucle principale du programme. Cette boucle contrôle le déplacement de l'envahisseur et teste l'état de la touche de tir.

2. **INVPOS**(position de l'envahisseur) : Deux octets sont utilisés pour mémoriser la position courante de l'envahisseur. A chaque déplacement, la nouvelle position est mémorisée.

3. **INVDIR** (direction de l'envahisseur) : Il s'agit d'une adresse unique qui contient la valeur 1 si le déplacement s'effectue de droite à gauche et la valeur 0 dans le cas contraire.
4. **GUNPOS** (position du fusil) : Deux octets sont utilisés afin de mémoriser la position courante du fusil. Lorsque le fusil est déplacé, cette variable est mise à jour.
5. **BULPOS** (position de la balle) : Deux octets sont utilisés pour mémoriser la position courante de la balle. Cette variable est mise à jour à chaque déplacement de la balle.

SOUS-PROGRAMME D'INITIALISATION (SETUP)

Voici la description des opérations réalisées par SETUP :

- Positionne l'envahisseur (INVPOS) en rangée 1 et colonne 1.
- Initialise à 1 la direction de l'envahisseur.
- Initialise la position du fusil (GUNPOS) en rangée 25 colonne 10.
- Initialise la position de la balle (BULPOS) en rangée 255 colonne 255. Ces coordonnées n'existant pas réellement, cela signifie simplement que la balle n'a pas encore été tirée.
- Initialise les couleurs à leurs valeurs par défaut ; fond bleu, encre jaune. Cela est réalisé grâce à l'appel d'une routine placée à l'adresse 48128.
- Initialise le mode écran 0 en chargeant le registre A avec la valeur 0 et en appelant une routine située à l'adresse 48142.
- Affichage de l'envahisseur. Le registre H contient la coordonnée verticale de l'envahisseur et le registre L la coordonnée horizontale. Cela est réalisé par l'instruction LD HL,(INVPOS). La routine de positionnement du curseur est ensuite appelée à l'adresse 47989. Le caractère représentant l'envahisseur est affiché. Cela est réalisé en mettant dans le registre A le code du caractère représentant l'envahisseur ; la routine d'affichage d'un caractère est alors appelée à l'adresse 47962.
- Affichage du fusil de la même manière que celle de l'envahisseur à l'exception de la variable GUNPOS qui est utilisée pour le posi-

tionnement du curseur. Le caractère représentant le fusil a pour code 244.

Adresse de départ **30041**
Adresse finale **30097**
Total **5619**

```
2601      260 SETUP   LD      H,1
2E01      270         LD      L,1
221879    280         LD      (INVPOS),HL
3E01      290         LD      A,1
321E79    300         LD      (INVDIR),A
260A      310         LD      H,10
2E19      320         LD      L,25
221A79    330         LD      (GUNPOS),HL
26FF      340         LD      H,255
2EFF      350         LD      L,255
221C79    360         LD      (BULPOS),HL
CD00BC    370         CALL    48128
3E00      380         LD      A,0
CD0EBC    390         CALL    48142
2A1879    400         LD      HL,(INVPOS)
CD75BB    410         CALL    47989
3EE0      420         LD      A,224
CD5ABB    430         CALL    47962
2A1A79    440         LD      HL,(GUNPOS)
CD75BB    450         CALL    47989
3EF4      460         LD      A,244
CD5ABB    470         CALL    47962
C9        480         RET
```

La routine SETUP est testée à l'aide des deux lignes suivantes :

```
1000 CALL 30041
1005 GOTO 1005
```

La figure suivante montre la position de l'envahisseur et celle du fusil :

ENVAHISSEUR ↓

← FUSIL

SOUS-PROGRAMME DE DÉPLACEMENT DU FUSIL

Cette routine teste l'état des touches de déplacement du fusil ; elle détermine la direction de celui-ci. La touche A est utilisée pour les déplacements vers la gauche et la touche D pour les déplacements vers la droite. Voici la description complète des opérations effectuées par cette routine :

1. Le registre A contient la valeur 69 (code de la touche A), la routine située à l'adresse 47902 est ensuite appelée. Si la touche n'est pas tapée, le programme saute à la routine de test de la touche D.
2. Si la touche A a été tapée, il faut d'abord regarder si le fusil se trouve dans le coin inférieur gauche de l'écran. Si c'est le cas, il ne peut pas se déplacer plus à gauche et le programme retourne à la boucle principale. Ce test est effectué en chargeant le registre A avec la valeur correspondant à la coordonnée verticale du fusil. Le registre A est ensuite comparé à la valeur 1 grâce à l'instruction CP 1. Si la valeur 0 est retournée (c'est-à-dire que le fusil est placé en colonne 1), le programme retourne à la boucle principale.
3. Si le fusil ne se trouve pas dans le coin inférieur gauche de l'écran, on peut alors le déplacer d'une position vers la gauche. Les registres H et L contiennent déjà les coordonnées horizontale et verticale de l'étape précédente. Nous allons utiliser la routine de positionnement du curseur située à l'adresse 47989. Ensuite, le registre est chargé avec la valeur 32 et la routine d'affichage d'un caractère (adresse 47962) est appelée. La valeur 32 représente le code de la touche espace.
4. Maintenant, le fusil doit être déplacé d'une position vers la gauche. Cela est réalisé par l'instruction HL, (GUNPOS) ; la valeur du registre H qui représente la coordonnée verticale du fusil est décrémentée. La nouvelle valeur de HL est stockée dans la variable représentant la position du fusil avec l'instruction LD (GUNPOS),HL. La routine de positionnement du curseur est appelée afin qu'il se trouve au niveau de la première position à gauche du fusil. Le registre A est alors chargé avec la valeur 244 (code correspondant au caractère représentant le fusil), puis la routine d'affichage d'un caractère est appelée à

nouveau. Le fusil a été déplacé d'une coordonnée vers la gauche et le programme retourne à la boucle principale.

5. Si, lors de l'exécution de la routine de déplacement du fusil, la touche A n'a pas été tapée, le programme effectue un branchement ici afin de tester l'état de la touche de déplacement vers la droite. Si elle n'est pas activée, le programme retourne à la boucle principale.

6. La routine de déplacement vers la droite est très semblable à celle qui le déplace dans le sens inverse. Les différences essentielles se situent dans le test de positionnement dans le bord inférieur droit de l'écran (colonne 20) et l'incrémentation de la coordonnée verticale courante du fusil.

Adresse de départ **30311**
Adresse finale **30386**
Total **9294**

```
3E45      1540 MOVGUN LD     A,69
CD1EBB    1550        CALL   47902
CA8E76    1560        JP     Z,MOVGR
2A1A79    1570        LD     HL,(GUNPOS)
7C        1580        LD     A,H
FE01      1590        CP     1
C8        1600        RET    Z
CD75BB    1610        CALL   47989
3E20      1620        LD     A,32
CD5ABB    1630        CALL   47962
2A1A79    1640        LD     HL,(GUNPOS)
25        1650        DEC    H
221A79    1660        LD     (GUNPOS),HL
CD75BB    1670        CALL   47989
3EF4      1680        LD     A,244
CD5ABB    1690        CALL   47962
C9        1700        RET
3E3D      1710 MOVGR  LD     A,61
CD1EBB    1720        CALL   47902
C8        1730        RET    Z
2A1A79    1740        LD     HL,(GUNPOS)
7C        1750        LD     A,H
FE14      1760        CP     20
C8        1770        RET    Z
```

```
CD75BB   1780          CALL   47989
3E20     1790          LD     A,32
CD5ABB   1800          CALL   47962
2A1A79   1810          LD     HL,(GUNPOS)
24       1820          INC    H
221A79   1830          LD     (GUNPOS),HL
CD75BB   1840          CALL   47989
3EF4     1850          LD     A,244
CD5ABB   1860          CALL   47962
C9       1870          RET
```

Cette routine est testée grâce aux trois lignes suivantes :

```
1000 CALL 30041
1005 CALL 30311
1010 GOTO 1005
```

Après avoir lancé le programme, la frappe de la touche A ou de la touche D déplace le fusil soit vers la gauche, soit vers la droite. On interrompt le programme en papant deux fois la touche ESC.

SOUS-PROGRAMME DE TEMPORISATION

Cette routine est analogue à celle décrite dans le chapitre précédent et ne nécessite donc pas d'explications supplémentaires.

Adresse de départ **30395**
Adresse finale **30403**
Total **1217**

```
0D       1960 DELAY    DEC    C
C2BB76   1970          JP     NZ,DELAY
05       1980          DEC    B
C2BB76   1990          JP     NZ,DELAY
C9       2000          RET
```

On peut tester cette routine à l'aide de l'instruction CALL 30395. On remarquera qu'il se passe un certain temps avant que le curseur et l'indicatif READY n'apparaissent à l'écran.

SOUS-PROGRAMME DE DÉPLACEMENT DE L'ENVAHISSEUR

Cette routine est très semblable à celle qui gère le déplacement du fusil. L'envahisseur se déplace à gauche ou à droite ; on affiche un caractère ayant pour code celui de l'espace, puis la coordonnée verticale courante est soit incrémentée, soit décrémentée. L'envahisseur est alors affiché à sa nouvelle position. Voici une description complète des opérations effectuées par cette routine :

1. On efface l'envahisseur en positionnant le curseur à l'endroit exact où il apparaît, puis on affiche un espace.
2. À ce niveau, on regarde si l'envahisseur n'a pas été touché par une balle. Cela est réalisé en comparant les coordonnées horizontale et verticale correspondant à la position de l'envahisseur et celles de la balle ; si elles sont identiques, le programme effectue un branchement à la routine HITHIM (celle-ci sera décrite plus tard).
3. Si l'envahisseur n'a pas été touché, il faut décider de sa direction. Cela est réalisé en testant la variable indiquant la direction de l'envahisseur (INVDIR). Si cette variable a la valeur 1, l'envahisseur se déplace de gauche à droite ; si elle a la valeur 0, il se déplace de droite à gauche.
4. Supposons que l'envahisseur se déplace vers la droite ; cette procédure est identique à celle qui gère le déplacement du fusil vers la droite. Il y a cependant une différence importante dans le fait que, lorsque le bord inférieur droit de l'écran est atteint, la variable INVDIR est mise à 0 afin que lorsqu'un nouvel appel est fait à la routine de déplacement de l'envahisseur, un déplacement vers la gauche soit effectué. La routine de déplacement de l'envahisseur vers la gauche est identique à celle qui gère le déplacement du fusil vers la gauche. Cependant, lorsque le coin inférieur gauche de l'écran est atteint, la variable INVDIR est mise à 1.

Adresse de départ 30107
Adresse finale 30218
Total 14210

```
2A1879    580 MOVINV  LD     HL,(INVPOS)
CD75BB    590         CALL   47989
3E20      600         LD     A,32
CD5ABB    610         CALL   47962
2A1879    620         LD     HL,(INVPOS)
ED5B1C79  630         LD     DE,(BULPOS)
A7        640         AND    A
ED52      650         SBC    HL,DE
CAF375    660         JP     Z,HITHIM
3A1E79    670         LD     A,(INVDIR)
FE01      680         CP     1
CAD775    690         JP     Z,MOVRGT
2A1879    700         LD     HL,(INVPOS)
7C        710         LD     A,H
FE01      720         CP     1
C2CA75    730         JP     NZ,LEFT
3E01      740         LD     A,1
321E79    750         LD     (INVDIR),A
C9        760         RET
25        770 LEFT    DEC    H
221879    780         LD     (INVPOS),HL
CD75BB    790         CALL   47989
3EE0      800         LD     A,224
CD5ABB    810         CALL   47962
C9        820         RET
2A1879    830 MOVRGT  LD     HL,(INVPOS)
7C        840         LD     A,H
FE14      850         CP     20
C2E675    860         JP     NZ,RIGHT
3E00      870         LD     A,0
321E79    880         LD     (INVDIR),A
C9        890         RET
24        900 RIGHT   INC    H
221879    910         LD     (INVPOS),HL
CD75BB    920         CALL   47989
3EE0      930         LD     A,224
CD5ABB    940         CALL   47962
C9        950         RET
2A1879    960 HITHIM  LD     HL,(INVPOS)
CD75BB    970         CALL   47989
```

```
3EE1      980      LD      A,225
CD5ABB    990      CALL    47962
01FFFF    1000     LD      BC,65535
CDBB76    1010     CALL    DELAY
CDBB76    1020     CALL    DELAY
CDBB76    1030     CALL    DELAY
C9        1040     RET
```

SOUS-PROGRAMME DE DÉPLACEMENT DE LA BALLE

Comme nous allons le voir au cours de sa description, cette routine ne réalise pas un simple déplacement de la balle :

1. Si l'on se réfère à la routine SETUP, on s'aperçoit que la position initiale de la balle correspond à la valeur 255 ; cela signifie que la balle n'a pas encore été tirée. Cette routine détermine donc tout d'abord la position de la balle ; si celle-ci correspond à la valeur 255, le programme effectue un branchement à la routine de test de la touche de tir.
2. Si la balle a été tirée, elle doit être déplacée sur l'écran ; il faut la déplacer d'une position vers le haut. Les registres H et L contenant déjà les coordonnées correspondant à la position de la balle, il suffit d'appeler la routine de positionnement du curseur, de charger le registre A avec la valeur 32 puis d'appeler la routine d'affichage d'un caractère afin d'effacer la balle.
3. Ensuite, le registre L est décrémenté afin de déplacer le curseur d'une position vers le haut mais aussi pour voir si la balle n'a pas atteint le sommet de l'écran. Supposons que la balle était affichée sur la première rangée, le registre L est alors décrémenté. L'instruction qui suit DEC L est JP NZ,MOVBL1 ; cette routine permet de réafficher la balle. Si, lorsqu'il est décrémenté, le registre L contient la valeur 0, on sait que la balle a atteint le sommet de l'écran et qu'il n'est pas nécessaire de la réafficher. Les coordonnées de la balle sont alors initialisées à 255 ; cela indique que la balle est immobile. Le programme retourne alors à la boucle principale.
4. Si la balle n'a pas atteint le sommet de l'écran, il suffit de l'afficher à sa nouvelle position. Le positionnement du curseur et

l'appel de la routine d'affichage d'un caractère réalisent cette tâche. Le programme retourne ensuite à la boucle principale.

5. Le programme se branche à cet endroit si la coordonnée verticale de la balle a pour valeur 255 dans la première étape. La touche de tir est ensuite testée ; on charge le registre A avec la valeur 18 (qui est le code de la touche ENTER), puis on appelle la routine de test des touches du clavier à l'adresse 47902. Si la touche n'a pas été tapée, le programme se branche sur une petite routine appelée SLOW1. Cette petite routine est une boucle de temporisation qui compense le fait que le programme n'a rien à faire si la touche n'a pas été tapée.

6. Si la touche ENTER a été tapée, la balle s'affiche juste au sommet du fusil. On charge le registre HL avec la position courante du fusil, on décrémente le registre L, puis on charge la variable BULPOS avec la valeur de HL. La balle est ensuite affichée avec la routine d'affichage d'un caractère. Le programme retourne alors à la boucle principale.

Adresse de départ **30227**
Adresse finale **30301**
Total **9078**

```
2A1C79   1130 MOVBUL LD    HL,(BULPOS)
7C       1140        LD    A,H
FEFF     1150        CP    255
CA3F76   1160        JP    Z,TRYBUL
CD75BB   1170        CALL  47989
3E20     1180        LD    A,32
CD5ABB   1190        CALL  47962
2A1C79   1200        LD    HL,(BULPOS)
2D       1210        DEC   L
C23376   1220        JP    NZ,MOVBL1
26FF     1230        LD    H,255
2EFF     1240        LD    L,255
221C79   1250        LD    (BULPOS),HL
C9       1260        RET
221C79   1270 MOVBL1 LD    (BULPOS),HL
CD75BB   1280        CALL  47989
3EEF     1290        LD    A,239
CD5ABB   1300        CALL  47962
C9       1310        RET
3E12     1320 TRYBUL LD    A,18
```

```
CD1EBB   1330          CALL   47902
CA5776   1340          JP     Z,SLOW1
2A1A79   1350          LD     HL,(GUNPOS)
2D       1360          DEC    L
221C79   1370          LD     (BULPOS),HL
CD75BB   1380          CALL   47989
3EEF     1390          LD     A,239
CD5ABB   1400          CALL   47962
C9       1410          RET
010002   1420 SLOW1    LD     BC,512
CDBB76   1430          CALL   DELAY
C9       1440          RET
```

Cette routine est testée à l'aide des lignes :

```
1000 CALL
1005 CALL
1010 GOTO 1005
```

On lance ce programme puis lorsque la touche ENTER est tapée la balle se déplace d'une position vers le haut.

BOUCLE PRINCIPALE

Cette dernière routine gère les appels successifs aux différentes parties du programme. Voici son action :

1. Lorsque le jeu commence, la routine SETUP est d'abord exécutée (CALL 30000).
2. La tâche de la boucle principale consiste en trois appels :

```
CALL MOVE INVADER
CALL MOVE BULLET
CALL MOVE GUN
CALL MOVE BULLET
```

   La routine de déplacement de la balle est appelée deux fois dans la boucle principale. Elle se déplace donc deux fois plus vite que le fusil et l'envahisseur.
3. Un test est effectué afin de voir si la touche X a été tapée ; si

c'est le cas, un retour au BASIC est réalisé. La routine située à l'adresse 42075 effectue cette opération.

4. Si la touche X n'a pas été tapée, le registre BC est chargé avec la valeur 4000 puis la routine de temporisation est appelée. Ensuite le programme retourne au début de la boucle principale.

Adresse de départ 30000
Adresse efinale 30031
Total 3978

```
CD5975    60 START   CALL   SETUP
CD9B75    70 LOOP    CALL   MOVINV
CD1376    80         CALL   MOVBUL
CD6776    90         CALL   MOVGUN
CD1376   100         CALL   MOVBUL
3E3F     110         LD     A,63
CD1EBB   120         CALL   47902
C203BB   130         JP     NZ,47875
01A00F   140         LD     BC,4000
CDBB76   150         CALL   DELAY
C33375   160         JP     LOOP
```

La totalité du programme peut désormais être testée ; on tape :

```
1000 CALL 30000
1005 STOP
```

Voici, pour ceux qui n'ont pas le temps d'entrer la totalité du programme en une seule fois, la façon de sauvegarder les routines. Pour la routine SETUP, par exemple :

1. On arrête le programme ENTHEX en tapant Q.
2. SAVE "SETUP",B,30041,57

   B signifie binaire, 30041 est l'adresse de départ et 57 la longueur de la routine calculée en soustrayant l'adresse de départ de l'adresse finale et en ajoutant 1.
3. Pour continuer, il suffit de charger le programme ENTHEX, de recharger la routine et de taper la suivante.

5. SOUS-PROGRAMMES UTILITAIRES

Fonction Déplace une ligne de texte vers la gauche à l'écran.

Remarques Le numéro de la ligne est spécifié après l'instruction CALL. Pour déplacer la ligne 13, on tape CALL 30250,13. En mode 2, la routine déplace la rangée d'un caractère vers la gauche. En mode 1, la routine doit être appelée deux fois afin de déplacer la ligne d'un caractère. En mode 0, la routine doit être appelée quatre fois.

Adresse de départ **30250**
Adresse finale **30299**
Total **6817**

```
DD6E00                 LD     L, (IX+0)
2600                   LD     H,0
CD1ABC                 CALL   48154
E5          LOOP 1     PUSH   HL
7E                     LD     A, (HL)
F5                     PUSH   AF
54                     LD     D,H
5D                     LD     E,L
EB                     EX     DE,HL
CD5676                 CALL   INCPTR
0E4F                   LD     C,79
7E          LOOP 2     LD     A, (HL)
12                     LD     (DE) ,A
CD5676                 CALL   INCPTR
EB                     EX     DE,HL
CD5676                 CALL   INCPTR
EB                     EX     DE,HL
0D                     DEC    C
20F3                   JR     NZ,LOOP2
F1                     POP    AF
12                     LD     (DE),A
E1                     POP    HL
7C                     LD     A,H
C608                   ADD    A,8
67                     LD     H,A
E638                   AND    56
20DD                   JR     NZ,LOOP1
C9                     RET
23          INCPTR     INC    HL
```

```
CBF4      310          SET    6,H
CBFC      320          SET    7,H
C9        330          RET
```

Fonction Déplace une ligne de texte vers la droite.

Remarques Cette routine est identique à la précédente, mais son effet est inverse. En mode 0, pour déplacer la ligne 22 d'un caractère vers la droite, on tape :

CALL 30350,22
CALL 30350,22
CALL 30350,22
CALL 30350,22

Adresse de départ **30350**
Adresse finale **30406**
Total **7903**

```
DD6E00    20           LD     L, (IX+0)
2600      30           LD     H,0
CD1ABC    40           CALL   48154
014F00    50           LD     BL,79
09        60           ADD    HL,BC
CBF4      70           SET    6,H
CBFC      80           SET    7,H
E5        90 LOOP 1    PUSH   HL
7E        100          LD     A, (HL)
F5        110          PUSH   AF
54        120          LD     D,H
5D        130          LD     E,L
CDC176    150          CALL   DECPTR
0E4F      160          LD     C,79
7E        170 LOOP 2   LD     A, (HL)
12        180          LD     (DE) ,A
CDC176    190          CALL   DECPTR
EB        200          EX     DE,HL
CDC176    210          CALL   DECPTR
EB        220          EX     DE,HL
0D        230          DEC    C
20F3      240          JR     NZ,LOOP2
```

```
F1      250        POP   AF
12      260        LD    (DE),A
E1      270        POP   HL
7C      280        LD    A,H
C608    290        ADD   A,8
67      300        LD    H,A
E638    310        ADD   56
20DE    320        JR    NZ,LOOP1
C9      330        RET
2B      340 DECPTR DEC   HL
CBF4    350        SET   6,H
CBFC    360        SET   7,H
C9      370        RET
```

Fonction Génère le son d'un canon laser.

Remarques Cette routine est appelée par l'instruction CALL 30000.

Adresse de départ 30000
Adresse finale 30069
Total 7089

```
3E08     20        LD    A,8
0E0F     30        LD    C,15
CD34BD   40        CALL  48436
3E07     50        LD    A,7
0E3E     60        LD    C,62
CD34BD   70        CALL  48436
3E00     80        LD    A,0
0E6E     90        LD    C,110
CD34BD  100        CALL  48436
3E01    110        LD    A,1
0E00    120        LD    C,0
CD34BD  130        CALL  48436
0E32    140        LD    C,50
3E00    150 LOOP   LD    A,0
C5      155        PUSH  BC
CD34BD  160        CALL  48436
3EC8    170        LD    A,200
F5      180 DELAY  PUSH  AF
3E1E    190        LD    A,30
3D      200 DEL    DEC   A
```

```
C25975   210          JP     NZ,DEL
F1       220          POP    AF
3D       230          DEC    A
C25675   240          JP     NZ,DELAY
C1       245          POP    BC
79       250          LD     A,C
C606     260          ADD    A,6
CA6E75   270          JP     Z,END
3D       280          DEC    A
4F       290          LD     C,A
C34E75   300          JP     LOOP
3E07     310 END      LD     A,7
0E3F     320          LD     C,63
CD34BD   330          CALL   48436
C9       340          RET
```

Fonction Génère le bruit de l'explosion d'une bombe.

Remarques Cette routine est mise en œuvre par l'instruction CALL 30100.

Adresse de départ **30100**
Adresse finale **30216**
Total **14897**

```
3E07      20          LD     A,7
0E3E      30          LD     C,62
CD34BD    40          CALL   48436
3E08      50          LD     A,8
0E0F      60          LD     C,15
CD34BD    70          CALL   48436
0E28      80          LD     C,40
3E00      90 LOOP     LD     A,0
C5       100          PUSH   BC
CD34BD   110          CALL   48436
3E0A     120          LD     A,10
F5       130 DELAY    PUSH   AF
3EFF     140          LD     A,255
3D       150 DEL      DEC    A
C2AF75   160          JP     NZ,DEL
F1       170          POP    AF
3D       180          DEC    A
C2AC75   190          JP     NZ,DELAY
```

```
C1      200         POP   BC
79      210         LD    A,C
D6C8    220         SUB   200
CAC575  230         JP    Z,NEXT
C6C9    240         ADD   A,201
4F      250         LD    C,A
C3A475  260         JP    LOOP
3E00    270 NEXT    LD    A,0
0E00    280         LD    C,0
CD34BD  290         CALL  48436
3E07    300         LD    A,7
0E37    310         LD    C,55
CD34BD  320         CALL  48436
3E00    330         LD    A,0
F5      340 NEXT1   PUSH  AF
4F      350         LD    C,A
CD34BD  360         CALL  48436
3E32    370         LD    A,50
F5      380 STEVE   PUSH  AF
3EFF    390         LD    A,255
3D      400 TRACEY  DEC   A
C2DF75  410         JP    NZ,TRACEY
F1      420         POP   AF
3D      430         DEC   A
C2DC75  440         JP    NZ,STEVE
F1      450         POP   AF
D61F    460         SUB   31
CAF375  470         JP    Z,LDEL
C620    480         ADD   A,32
C3D575  490         JP    NEXT1
3E64    500 LDEL    LD    A,100
F5      510 LDEL1   PUSH  AF
3EFF    520         LD    A,255
3D      530 LDEL2   DEC   A
C2F875  540         JP    NZ,LDEL2
F1      550         POP   AF
3D      560         DEC   A
C2F575  570         JP    NZ,LDEL1
3E07    580         LD    A,7
0E3F    590         LD    C,63
CD34BD  600         CALL  48436
C9      610         RET
```

Codes machine du microprocesseur Z80

ADC A, (HL)	8E	AND A	A7
ADC A, (IX + d)	DD8Ed	AND B	A0
ADC A, (IY + d)	FD8Ed	AND C	A1
ADC A, A	8F	AND D	A2
ADC A, B	88	AND E	A3
ADC A, C	89	AND H	A4
ADC A, D	8A	AND L	A5
ADC A, E	8B	AND n	E6n
ADC A, H	8C	BIT 0, (HL)	CB46
ADC A, L	8D	BIT 0, (IX + D)	DDCBd46
ADC A, n	CEn	BIT 0, (IY + d)	FDCBd46
ADC HL, BC	ED4A	BIT 0, A	CB47
ADC HL, DE	ED5A	BIT 0, B	CB40
ADC HL, HL	ED6A	BIT 0, C	CB41
ADC HL, SP	ED7A	BIT 0, D	CB42
ADD A, (HL)	86	BIT 0, E	CB43
ADD A, (IX + d)	DD86d	BIT 0, H	CB44
ADD A, (IY + d)	FD86d	BIT 0, L	CB45
ADD A, A	87	BIT 1, (HL)	CB4E
ADD A, B	80	BIT 1, (IX + d)	DDCBd4E
ADD A, C	81	BIT 1, (IY + d)	FDCBd4E
ADD A, D	82	BIT 1, A	CB4F
ADD A, E	83	BIT 1, B	CB48
ADD A, H	84	BIT 1, C	CB49
ADD A, L	85	BIT 1, D	CB4A
ADD A, n	C6n	BIT 1, E	CB4B
ADD HL, BC	09	BIT 1, H	CB4C
ADD HL, DE	19	BIT 1, L	CB4D
ADD HL, HL	29	BIT 2, (HL)	CB56
ADD HL, SP	39	BIT 2, (IX + d)	DDCBd56
ADD IX, BC	DD09	BIT 2, (IY + d)	FDCBd56
ADD IX, DE	DD19	BIT 2, A	CB57
ADD IX, IX	DD29	BIT 2, B	CB50
ADD IX, SP	DD39	BIT 2, C	CB51
ADD IY, BC	FD09	BIT 2, D	CB52
ADD IY, DE	FD19	BIT 2, E	CB53
ADD IY, IY	FD29	BIT 2, H	CB54
ADD IY, SP	FD39	BIT 2, L	CB55
AND(HL)	A6	BIT 3, (HL)	CB5E
AND (IX + d)	DDA6d	BIT 3, (IX + d)	DDCBd5E
AND (IY + d)	FDA6d	BIT 3, (IY + d)	FDCBd5E
BIT 3, A	CB5F	CALL nn	CDnn
BIT 3, B	CB58	CALL NZ, nn	C4nn

BIT 3, C	CB59
BIT 3, D	CB5A
BIT 3, E	CB5B
BIT 3, H	CB5C
BIT 3, L	CB5D
BIT 4, (HL)	CB66
BIT 4, (IX + d)	DDCBd66
BIT 4, (IY + d)	FDCBd66
BIT 4, A	CB67
BIT 4, B	CB60
BIT 4, C	CB61
BIT 4, D	CB62
BIT 4, E	CB63
BIT 4, H	CB64
BIT 4, L	CB65
BIT 5, (HL)	CB6E
BIT 5, (IX + d)	DDCBd6E
BIT 5, (IY + D)	FDCBd6E
BIT 5, A	CB6F
BIT 5, B	CB68
BIT 5, C	CB69
BIT 5, D	CB6A
BIT 5, E	CB6B
BIT 5, H	CB6C
BIT 5, L	CB6D
BIT 6, (HL)	CB76
BIT 6, (IX + d)	DDCBd76
BIT 6, (IY + d)	FDCBd76
BIT 6, A	C877
BIT 6, B	CB70
BIT 6, C	CB71
BIT 6, D	CB72
BIT 6, E	CB73
BIT 6, H	CB74
BIT 6, L	CB75
BIT 7, (HL)	CB7E
BIT 7, (IX + d)	DDCBd7E
BIT 7, (IY + d)	FDCBd7E
BIT 7, A	CB7F
BIT 7, B	CB78
BIT 7, C	CB79
BIT 7, D	CB7A
BIT 7, E	CB7B
BIT 7, H	CB7C
BIT 7, L	CB7D
CALL C, nn	DCnn
CALL M, nn	FCnn
CALL NC, nn	D4nn
IM 0	ED46

CALL P, nn	F4nn
CALL PE, nn	ECnn
CALL PO, nn	E4nn
CALL Z, nn	CCnn
CCF	3F
CP (HL)	BE
CP (IX + d)	DDBEd
CP (IY + d)	FDBEd
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP n	FEn
CPD	EDA9
CPDR	EDB9
CPI	EDA1
CPIR	EDB1
CPL	2F
DAA	27
DEC (HL)	35
DEC (IX + d)	DD35d
DEC (IY + d)	FD35d
DEC A	3D
DEC B	05
DEC BC	0B
DEC C	0D
DEC D	15
DEC DE	1B
DEC E	1D
DEC H	25
DEC HL	2B
DEC IX	DD2B
DEC IY	FD2B
DEC L	2D
DEC SP	3B
DI	F3
DJNZ, d	10d
E1	FB
EX (SP), HL	E3
EX (SP), IX	DDE3
EX (SP), IY	FDE3
EX AF, AF	08
EX DE, HL	EB
EXX	D9
HALT	76
LD (HL), A	77

IM 1	ED56	LD (HL), B	70
IM 2	ED5E	LD (HL), C	71
IN A, (C)	ED78	LD (HL), D	72
IN A, (n)	DBn	LD (HL), E	73
IN B, (C)	ED40	LD (HL), H	74
IN C, (C)	ED48	LD (HL), L	75
IN D, (C)	ED50	LD (HL), n	36n
IN E, (C)	ED58	LD (IX + d), A	DD77d
IN H, (C)	ED60	LD (IX + d), B	DD70d
IN L, (C)	ED68	LD (IX + d), C	DD71d
INC (HL)	34	LD (IX + d), D	DD72d
INC (IX + d)	DD34d	LD (IX + d), E	DD73d
INC (IY + d)	FD34d	LD (IX + d), H	DD74d
INC A	3C	LD (IX + d), L	DD75d
INC B	04	LD (IX + d), n	DD36dn
INC BC	03	LD (IY + d), A	FD77d
INC C	0C	LD (IY + d), B	FD70d
INC D	14	LD (IY + d), C	FD71d
INC DE	13	LD (IY + d), D	FD72d
INC E	1C	LD (IY + d), E	FD73d
INC H	24	LD (IY + d), H	FD74d
INC HL	23	LD (IY + d), L	FD75d
INC IX	DD23	LD (IY + d), n	FD36dn
INC IY	FD23	LD (nn), A	32nn
INC L	2C	LD (nn), BC	ED43nn
INC SP	33	LD (nn), DE	ED53nn
IND	EDAA	LD (nn), HL	22nn
INDR	EDBA	LD (nn), IX	DD22nn
INI	EDA2	LD (nn), IY	FD22nn
INIR	EDB2	LD (nn), SP	ED73nn
JP (HL)	E9	LD A, (BC)	0A
JP (IX)	DDE9	LD A, (DE)	1A
JP (IY)	FDE9	LD A, (HL)	7E
JP C, nn	DAnn	LD A, (IX + d)	DD7Ed
JP M, nn	FAnn	LD A, (IY + d)	FD7Ed
JP NC, nn	D2nn	LD A, (nn)	3Ann
JP nn	C3nn	LD A, A	7F
JP NZ, nn	C2nn	LD A, B	78
JP P, nn	F2nn	LD A, C	79
JP PE, nn	EAnn	LD A, D	7A
JP PO, nn	E2nn	LD A, E	7B
JP Z, nn	CAnn	LD A, H	7C
JR C, d	38d	LD A, I	ED57
JR, d	18d	LD A, L	7D
JR NC, d	30d	LD A, n	3En
JR NZ, d	20d	LD B, (HL)	46
JR Z, d	28d	LD B, (IX + d)	DD46d
LD (BC), A	02	LD B, (IY + d)	FD46d
LD (DE), A	12	LD B, A	47

LD B, B	40
LD B, C	41
LD B, D	42
LD B, E	43
LD B, H	44
LD B, L	45
LD B, n	06n
LD B, C (nn)	ED4Bnn
LD BC (nn)	01nn
LD C, (HL)	4E
LD C, (IX + d)	DD4Ed
LD C, (IY + d)	FD4Ed
LD C, A	4F
LD C, B	48
LD C, C	49
LD C, D	4A
LD C, E	4B
LD C, H	4C
LD C, L	4D
LD C, n	0En
LD D, (HL)	56
LD D, (IX + d)	DD56d
LD D, (IY + d)	FD56d
LD D, A	57
LD D, B	50
LD D, C	51
LD D, D	52
LD D, E	53
LD D, H	54
LD D, L	55
LD D, n	16n
LD DE, (nn)	ED58nn
LD DE, nn	11nn
LD E, (HL)	5E
LD E, (IX + d)	DD5Ed
LD E, (IY + d)	FD5Ed
LD E, A	5F
LD E, B	58
LD E, C	59
LD E, D	5A
LD E, E	5B
LD E, H	5C
LD E, L	5D
LD E, n	1En
LD H, (HL)	66
LD H, (IX + d)	DD66d
LD H, (IY + d)	FF66d
LD H, A	67
LD H, B	60
LD H, D	62
LD H, E	63
LD H, H	64
LD H, L	65
LD H, n	26n
LD HL, (nn)	2Ann
LD HL, nn	21nn
LD I, A	ED47
LD IX, (nn)	DD2Ann
LD IX, nn	DD21nn
LD IY, (nn)	FD2Ann
LD IY, nn	FD21nn
LD L, (HL)	6E
LD L, (IX + d)	DD6Ed
LD L, (IY + d)	FD6Ed
LD L, A	6F
LD L, B	68
LD L, C	69
LD L, D	6A
LD L, E	6B
LD L, H	6C
LD L, L	6D
LD L, n	2En
LD SP, (nn)	ED7Bnn
LD SP, HL	F9
LD SP, IX	DDF9
LD SP, IY	FDF9
LD SP, nn	31nn
LDD	EDA8
LDDR	EDB8
LDI	EDAO
LDIR	EDB0
NEG	ED44
NOP	00
OR (HL)	B6
OR (IX + d)	DDB6d
OR (IY + d)	FDB6d
OR A	B7
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR H	B4
OR L	B5
OR n	F6n
OTDR	EDBB
OTIR	EDB3
OUT (C), A	ED79
OUT (C), B	ED41

LD H, C	61
OUT (C), D	ED51
OUT (C), E	ED59
OUT (C), H	ED61
OUT (C), L	ED69
OUT (n), A	D3n
OUTD	EDAB
OUTI	EDA3
POPAF	F1
POP BC	C1
POP DE	D1
POP HL	E1
POP IX	DDE1
POP IY	FDE1
PUSH AF	F5
PUSH BC	C5
PUSH DE	D5
PUSH HL	E5
PUSH IX	DDE5
PUSH IY	FDE5
RES 0, (HL)	CB86
RES 0, (IX + d)	DDCBd86
RES 0, (IY + d)	FDCBd86
RES 0, A	CB87
RES 0, B	CB80
RES 0, C	CB81
RES 0, D	CB82
RES 0, E	CB83
RES 0, H	CB84
RES 0, L	CB85
RES 1, (HL)	CB8E
RES 1, (IX + d)	DDCBd8E
RES 1, (IY + d)	FDCBd8E
RES 1, A	CB8F
RES 1, B	CB88
RES 1, C	CB89
RES 1, D	CB8A
RES 1, E	CB8B
RES 1, H	CB8C
RES 1, L	CB8D
RES 2, (HL)	CB96
RES 2, (IX + d)	DDCBd96
RES 2, (IY + d)	FDCBd96
RES 2, A	CB97
RES 2, B	CB90
RES 2, C	CB91
RES 2, D	CB92
RES 2, E	CB93
RES 2, H	CB94
OUT (C), C	ED49
RES 3, (IX + d)	DDCBd9E
RES 3, (IY + d)	FDCBd9E
RES 3, A	CB9F
RES 3, B	CB98
RES 3, C	CB99
RES 3, D	CB9A
RES 3, E	CB9B
RES 3, H	CB9C
RES 3, L	CB9D
RES 4, (HL)	CBA6
RES 4, (IX + d)	DDCBdA6
RES 4, (IY + d)	FDCBdA6
RES 4, A	CBA7
RES 4, B	CBA0
RES 4, C	CBA1
RES 4, D	CBA2
RES 4, E	CBA3
RES 4, H	CBA4
RES 4, L	CBA5
RES 5, (HL)	CBAE
RES 5, (IX + d)	DDCBdAE
RES 5, (IY + d)	FDCBdAE
RES 5, A	CBAF
RES 5, B	CBA8
RES 5, C	CBA9
RES 5, D	CBAA
RES 5, E	CBAB
RES 5, H	CBAC
RES 5, L	CBAD
RES 6, (HL)	CBB6
RES 6, (IX + d)	DDCBdB6
RES 6, (IY + d)	FDCBdB6
RES 6, A	CBB7
RES 6, B	CBB0
RES 6, C	CBB1
RES 6, D	CBB2
RES 6, E	CBB3
RES 6, H	CBB4
RES 6, L	CBB5
RES 7, (HL)	CBBE
RES 7, (IX + d)	DDCBdBE
RES 7, (IY + d)	FDCBdBE
RES 7, A	CBBF
RES 7, B	CBB8
RES 7, C	CBB9
RES 7, D	CBBA
RES 7, E	CBBB
RES 7, H	CBBC

RES 2, L	CB95
RES 3, (HL)	CB9E
RET C	D8
RET M	F8
RET NC	D0
RET NZ	C0
RET P	F0
RET PE	E8
RET P0	E0
RET Z	C8
RETI	ED4D
RETN	ED45
RL (HL)	CB16
RL (IX + d)	DDCBd16
RL (IY + d)	FDCBd16
RL A	CB17
RL B	CB10
RL C	CB11
RL D	CB12
RL E	CB13
RL H	CB14
RL L	CB15
RLA	17
RLC (HL)	CB06
RLC (IX + d)	DDCBd06
RLC (IY + d)	FDCBd06
RLC A	CB07
RLC B	CB00
RLC C	CB01
RLC D	CB02
RLC E	CB03
RLC H	CB04
RLC L	CB05
RLCA	07
RLD	ED6F
RR (HL)	CB1E
RR (IX + d)	DDC8d1E
RR (IY + d)	FDCBd1E
RR A	CB1F
RR B	CB18
RR C	CB19
RR D	CB1A
RR E	CB1B
RR H	CB1C
RR L	CB1D
RRA	1F
RRC (HL)	CB0E
RRC (IX + d)	DDCBd0E
RRC (IY + d)	FDCBd0E
RES 7, L	CBBD
RET	C9
RRC D	CB0A
RRC E	CB0B
RRC H	CB0C
RRC L	CB0D
RRC A	0F
RRD	ED67
RST 0	C7
RST10H	D7
RST 18H	DF
RST 20H	E7
RST 28H	EF
RST 30H	F7
RST 38H	FF
RST 8	CF
SBC A, (HL)	9E
SBC A, (IX + d)	DD9Ed
SBC A, (IY + d)	FD9Ed
SBC A, A	9F
SBC A, B	98
SBC A, C	99
SBC A, D	9A
SBC A, E	9B
SBC A, H	9C
SBC A, L	9D
SBC A, n	DEn
SBC HL, BC	ED42
SBC HL, DE	ED52
SBC HL, HL	ED62
SBC HL, SP	ED72
SCF	37
SET 0, (HL)	CBC6
SET 0, (IX + d)	DDCBdC6
SET 0, (IY + d)	FDCBdC6
SET 0, A	CBC7
SET 0, B	CBC0
SET 0, C	CBC1
SET 0, D	CBC2
SET 0, E	CBC3
SET 0, H	CBC4
SET 0, L	CBC5
SET 1, (HL)	CBCE
SET 1, (IX + d)	DDCBdCE
SET 1, (IY + d)	FDCBdCE
SET 1, A	CBCF
SET 1, B	CBC8
SET 1, C	CBC9
SET 1, D	CBCA

RRC A	CB0F
RRC B	CB08
RRC C	CB09
SET 2, (HL)	CBD6
SET 2, (IX + d)	DDCBdD6
SET 2, (IY + d)	FDCBdD6
SET 2, A	CBD7
SET 2, B	CBD0
SET 2, C	CBD1
SET 2, D	CBD2
SET 2, E	CBD3
SET 2, H	CBD4
SET 2, L	CBD5
SET 3, (HL)	CBDE
SET 3, (IX + d)	DDCBdDE
SET 3, (IY + d)	FDCBdDE
SET 3, A	CBDF
SET 3, B	CBD8
SET 3, C	CBD9
SET 3, D	CBDA
SET 3, E	CBDB
SET 3, H	CBDC
SET 3, L	CBDD
SET 4, (HL)	CBE6
SET 4, (IX + d)	DDCbE6
SET 4, (IY + d)	FDCBdE6
SET 4, A	CBE7
SET 4, B	CBE0
SET 4, C	CBE1
SET 4, D	CBE2
SET 4, E	CBE2
SET 4, H	CBE4
SET 4, L	CBE5
SET 5, (HL)	CBEE
SET 5, (IX + d)	DDCBdEE
SET 5, (IY + d)	FDCBdEE
SET 5, A	CBEF
SET 5, B	CBE8
SET 5, C	CBE9
SET 5, D	CBEA
SET 5, E	CBEB
SET 5, H	CBEC
SET 5, L	CBED
SET 6, (HL)	CBF6
SET 6, (IX + d)	DDCBdF6
SET 6, (IY + d)	FDCBdF6
SET 6, A	CBF7
SET 6, B	CBF0
SET 6, C	CBF1

SET 1, E	CBCB
SET 1, H	CBCC
SET 1, L	CBCD
SET 7, (HL)	CBFE
SET 7, (IX + d)	DDCBdFE
SET 7, (IY + d)	FDCBdFE
SET 7, A	CBFF
SET 7, B	CBF8
SET 7, C	CBF9
SET 7, D	CBFA
SET 7, E	CBFB
SET 7, H	CBFC
SET 7, L	CBFD
SLA (HL)	CB26
SLA (IX + d)	DDCBd26
SLA (IY + d)	FDCBd26
SLA A	CB27
SLA B	CB20
SLA C	CB21
SLA D	CB22
SLA E	CB23
SLA H	CB24
SLA L	CB25
SRA (HL)	CB2E
SRA (IX + d)	DDCBd2E
SRA (IY + d)	FDCBd2E
SRA A	CB2F
SRA B	CB28
SRA C	CB29
SRA D	CB2A
SRA E	CB2B
SRA H	CB2C
SRA L	CB2D
SRL (HL)	CB3E
SRL (IX + d)	DDCBd3E
SRL (IY + d)	FDCBd3E
SRL A	CB3F
SRL B	CB38
SRL C	CB39
SRL D	CB3A
SRL E	CB3B
SRL H	CB3C
SRL L	CB3D
SUB (HL)	96
SUB (IX + d)	DD96d
SUB (IY + d)	FD96d
SUB A	97
SUB B	90
SUB C	91

SET 6, D	CBF2	SUB D	92
SET 6, E	CBF3	SUB E	93
SET 6, H	CBF4	SUB H	94
SET 6, L	CBF5	SUB L	95
SUB n	D6n	XOR C	A9
XOR (HL)	AE	XOR D	AA
XOR (IX + d)	DDAEd	XOR E	AB
XOR (IY + d)	FDAEd	XOR H	AC
XOR A	AF	XOR L	AD
XOR B	A8	XOR n	EEn

ANNEXE 2

Tableau de conversion hexadécimal/décimal

	0	1	2	3	4	5	6	7	8	9	OA	OB	OC	OD	OE	OF
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

ANNEXE 3
Le système binaire

Bien que la connaissance du système binaire ne soit pas indispensable à l'apprentissage de la programmation en langage machine, celle-ci peut se révéler utile à la résolution de certains problèmes particuliers. Le principe de la notation binaire est d'ailleurs relativement facile à comprendre. Dans le chapitre consacré au mode de stockage des nombres, il a été indiqué que chaque emplacement mémoire (c'est-à-dire chaque octet) pouvait recevoir un nombre dont la valeur décimale est comprise entre 0 et 255. Cela résulte de la définition même de l'octet. Un octet est formé de l'association de huit unités élémentaires appelées "bits" (le mot "bit" est la contraction de l'expression anglo-saxonne *binary digit* signifiant "chiffre binaire"). Les bits constituant un octet sont numérotés de la manière suivante :

Huit bits forment un octet	7	6	5	4	3	2	1	0

Ces huit bits peuvent être considérés comme représentant autant d'interrupteurs. Un interrupteur peut être ouvert ou fermé. Quand il est ouvert, le courant ne passe pas et le bit correspondant a pour valeur zéro. Quand il est fermé ou, par analogie, lorsque le courant passe, le bit correspondant a pour valeur 1 (on dit dans ce cas que le bit est "positionné"). Cependant, de même que dans le nombre décimal 33, par exemple, les deux chiffres 3 n'ont pas la même importance (il n'ont pas le même "poids"), le numéro du bit dans un octet détermine son poids dans la valeur donnée à l'octet. Le diagramme ci-dessous donne la valeur décimale d'un bit positionné dans un octet :

Valeur du bit	128	64	32	16	8	4	2	1
Numéro du bit	7	6	5	4	3	2	1	0

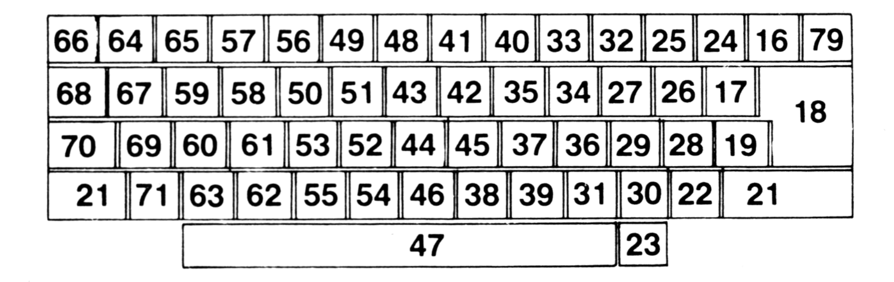
Ainsi, lorsque le bit 4 est positionné, il contribue pour 16 à la valeur décimale de l'octet. En d'autres termes, si les bits 1 et 3 d'un octet sont positionnés alors que tous les autres ne le sont pas, cet octet aura pour valeur décimale 10. En effet, le bit 1, lorsqu'il est positionné, correspond à 2 (décimal) et le bit 3 à 8. Lorsque tous les bits d'un octet sont positionnés, celui-ci a pour valeur décimale 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255. Lorsqu'au contraire tous les bits sont à zéro, l'octet vaut naturellement 0. C'est pourquoi il est possible de donner à un octet toute valeur décimale entière comprise entre 0 et 255. Lorsque l'on donne une telle valeur décimale à un octet, au moyen des instructions PEEK et POKE ou de leur équivalent en langage machine, l'ordinateur effectue lui-même la conversion entre les systèmes décimal et binaire, afin de savoir quels bits de l'octet doivent être positionnés pour représenter la valeur décimale que l'on veut stocker.

Si les bits 1, 4 et 6 d'un octet sont positionnés, quelle est la valeur décimale de cet octet ?

Quels bits doivent être positionnés pour que la valeur décimale d'un octet corresponde à 67 ?

ANNEXE 4
Codes des touches

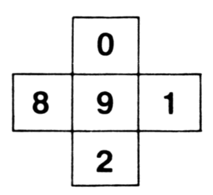
CLAVIER



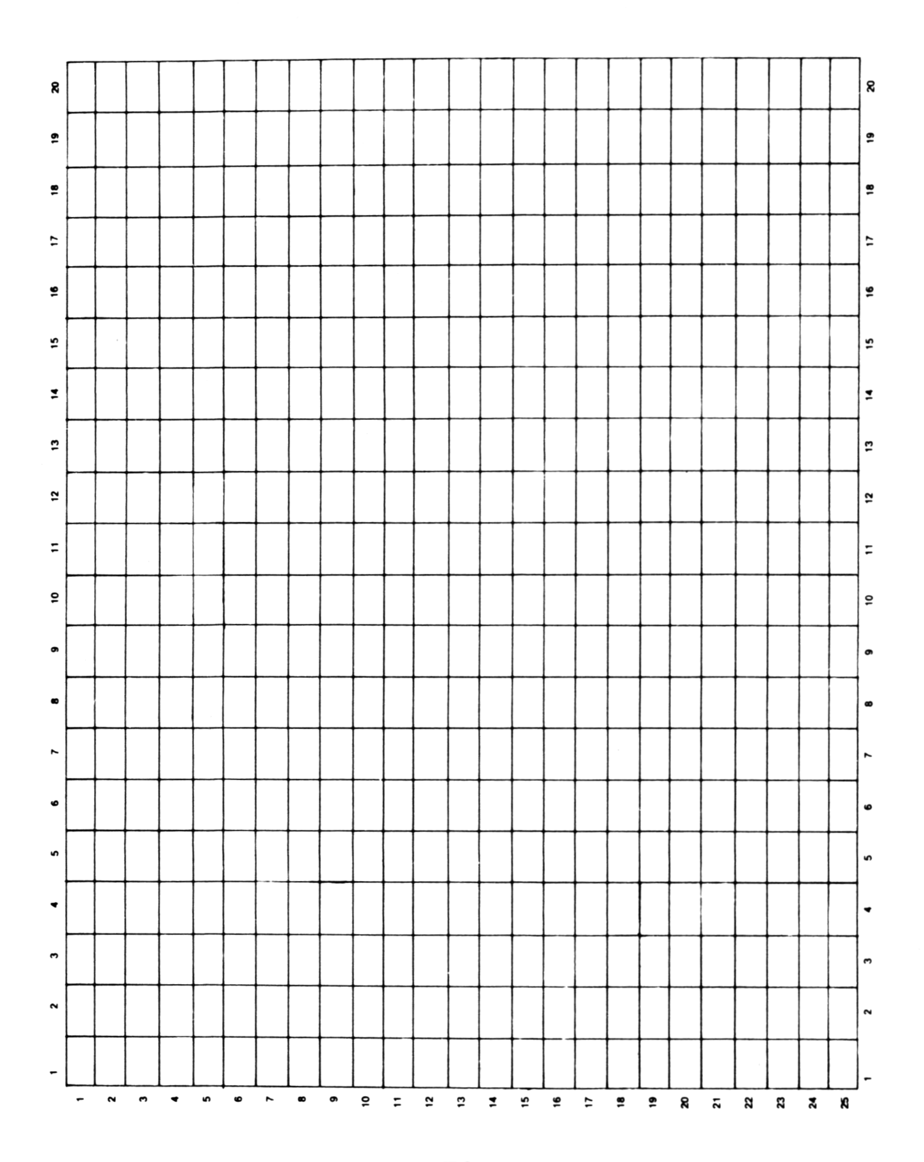
PAVÉ NUMÉRIQUE

10	11	3
20	12	4
13	14	5
15	7	6

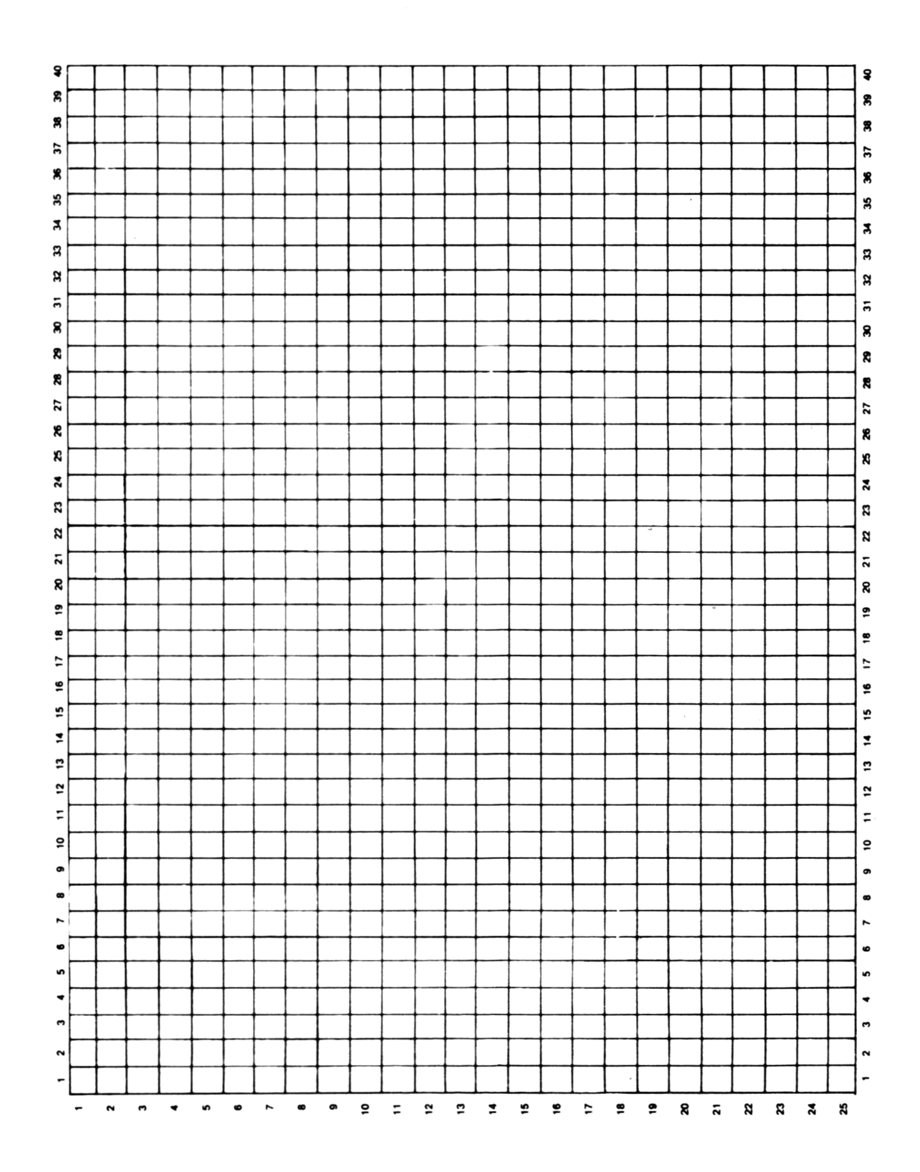
CURSEUR



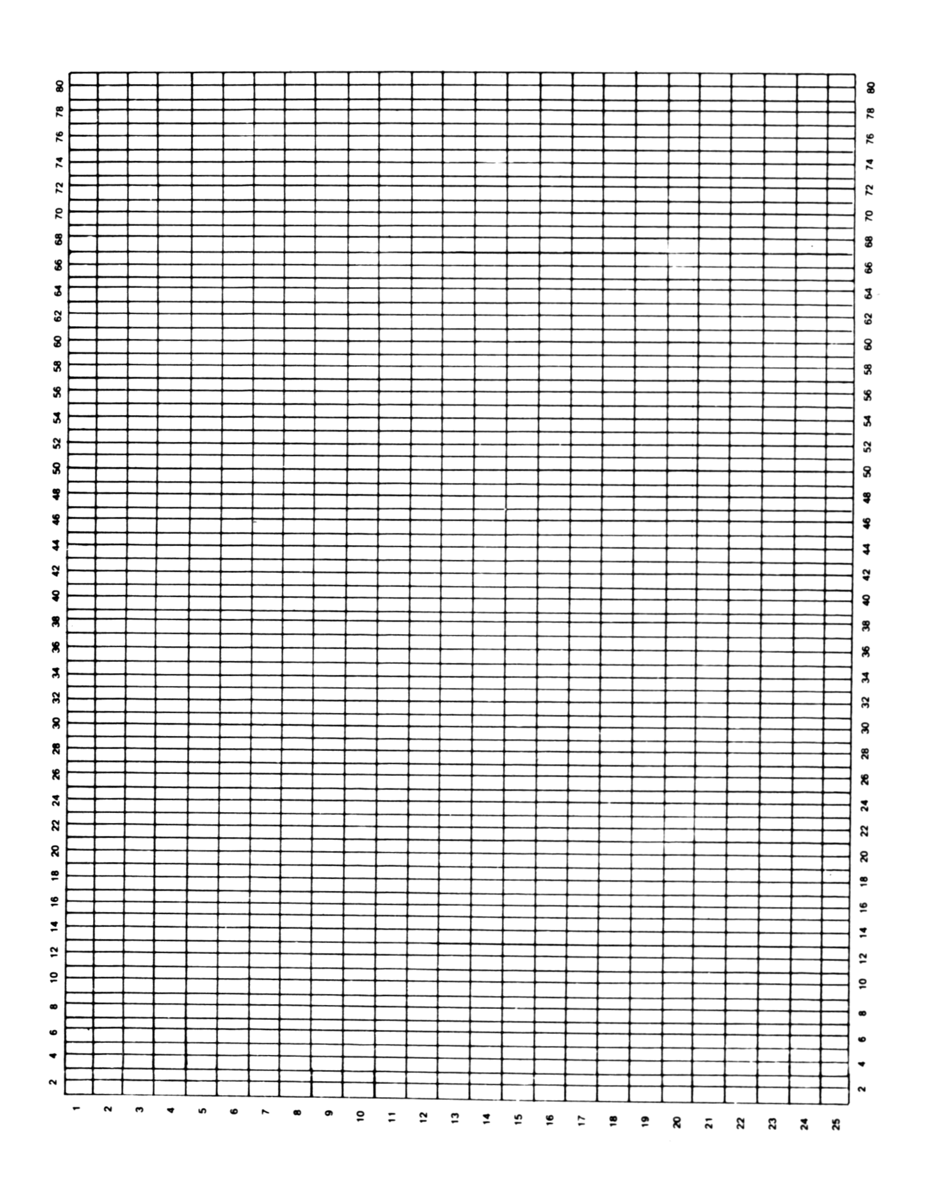
ANNEXE 5
Écran mode 0 (20 colonnes x 25 rangées)



Écran mode 1 (40 colonnes x 25 rangées)



Écran mode 2 (80 colonnes x 25 rangées)



ANNEXE 6
Générateur de caractères

Ce programme permet de redessiner les caractères dont le code est compris entre 32 et 255. Les nouveaux caractères pourront être utilisés dans des programmes personnels.

Adresse de départ **30000**
Adresse finale **30364**
Total **43837**

```
AF          20          XOR    A
327A76      30          LD     (CURCHR),A
327B76      40          LD     (CURLIN),A
327C76      50          LD     (CURCOL),A
3E01        60          LD     A,1
CD0EBC      70          CALL   48142
CDB975      80 LOOP     CALL   GRID
CDFA75      90          CALL   CURSOR
CD1776     100          CALL   CHRNUM
CD5576     110          CALL   DRCHAR
CD18BB     120          CALL   47896
FE0D       130          CP     13
C8         140          RET    Z
213F75     150          LD     HL,LOOP
E5         160          PUSH   HL
FEF0       170          CP     240
CA7975     180          JP     Z,CURUP
FEF1       190          CP     241
CA8375     200          JP     Z,CURDWN
FEF2       210          CP     242
CA8E75     220          JP     Z,CURLEF
FEF3       230          CP     243
CA9875     240          JP     Z,CURRIG
FE20       250          CP     " "
CAA375     260          JP     Z,SWITCH
FE4E       270          CP     "N"
CAA975     280          JP     Z,NEXT
FE50       290          CP     "P"
```

```
CAB175   300         JP    Z,PREV
C9       310         RET
3A7B76   320 CURUP   LD    A,(CURLIN)
A7       330         AND   A
C8       340         RET   Z
3D       350         DEC   A
327B76   360         LD    (CURLIN),A
C9       370         RET
3A7B76   380 CURDWN  LD    A,(CURLIN)
FE07     390         CP    7
C8       400         RET   Z
3C       410         INC   A
327B76   420         LD    (CURLIN),A
C9       430         RET
3A7C76   440 CURLEF  LD    A,(CURCOL)
A7       450         AND   A
C8       460         RET   Z
3D       470         DEC   A
327C76   480         LD    (CURCOL),A
C9       490         RET
3A7C76   500 CURRIG  LD    A,(CURCOL)
FE07     510         CP    7
C8       520         RET   Z
3C       530         INC   A
327C76   540         LD    (CURCOL),A
C9       550         RET
CD3B76   560 SWITCH  CALL  PIXPOS
AE       570         XOR   (HL)
77       580         LD    (HL),A
C9       590         RET
3A7A76   600 NEXT    LD    A,(CURCHR)
3C       610         INC   A
327A76   620         LD    (CURCHR),A
C9       630         RET
3A7A76   640 PREV    LD    A, (CURCHR)
3D       650         DEC   A
327A76   660         LD    (CURCHR),A
C9       670         RET
3A7A76   680 GRID    LD    A,(CURCHR)
6F       690         LD    L,A
2610     700         LD    H,16
29       710         ADD   HL,HL
29       720         ADD   HL,HL
29       730         ADD   HL,HL
EB       740         EX    DE,HL
2100C0   750         LD    HL,49152
```

```
0608     760         LD     B,8
C5       770 GRIDL1  PUSH   BC
D5       780         PUSH   DE
E5       790         PUSH   HL
1A       800         LD     A,(DE)
4F       810         LD     C,A
0608     820         LD     B,8
E5       830 GRIDL2  PUSH   HL
117D76   840         LD     DE,CHAR1
CB11     850         RL     C
3003     860         JR     NC,GRIDL3
118D76   870         LD     DE,CHAR2
1A       880 GRIDL3  LD     A,(DE)
77       890         LD     (HL),A
13       900         INC    DE
23       910         INC    HL
1A       920         LD     A,(DE)
77       930         LD     (HL),A
13       940         INC    DE
2B       950         DEC    HL
7C       960         LD     A,H
C608     970         ADD    A,8
67       980         LD     H,A
E638     990         AND    56
20F0     1000        JR     NZ,GRIDL3
E1       1010        POP    HL
23       1020        INC    HL
23       1030        INC    HL
10E0     1040        DJNZ   GRIDL2
E1       1050        POP    HL
115000   1060        LD     DE,80
19       1070        ADD    HL,DE
D1       1080        POP    DE
13       1090        INC    DE
C1       1100        POP    BC
10CF     1110        DJNZ   GRIDL1
C9       1120        RET
3A7B76   1130 CURSOR LD     A,(CURLIN)
3C       1140        INC    A
6F       1150        LD     L,A
3A7C76   1160        LD     A,(CURCOL)
3C       1170        INC    A
67       1180        LD     H,A
CD75BB   1190        CALL   47989
CD3B76   1200        CALL   PIXPOS
A6       1210        AND    (HL)
```

```
D601     1220          SUB    1
9F       1230          SBC    A,A
2F       1240          CPL
E676     1250          AND    118
EE9F     1260          XOR    159
CD5ABB   1270          CALL   47962
C9       1280          RET
21010C   1290 CHRNUM   LD     HL,3073
CD75BB   1300          CALL   47989
3A7A76   1310          LD     A,(CURCHR)
4F       1320          LD     C,A
0664     1330          LD     B,100
CD2D76   1340          CALL   CHRNL1
060A     1350          LD     B,10
CD2D76   1360          CALL   CHRNL1
0601     1370          LD     B,1
162F     1380 CHRNL1   LD     D,47
79       1390          LD     A,C
90       1400 CHRNL2   SUB    B
14       1410          INC    D
30FC     1420          JR     NC,CHRNL2
80       1430          ADD    A,B
4F       1440          LD     C,A
7A       1450          LD     A,D
CD5ABB   1460          CALL   47962
C9       1470          RET
3A7A76   1480 PIXPOS   LD     A,(CURCHR)
6F       1490          LD     L,A
2610     1500          LD     H,16
29       1510          ADD    HL,HL
29       1520          ADD    HL,HL
29       1530          ADD    HL,HL
3A7B76   1540          LD     A,(CURLIN)
85       1550          ADD    A,L
6F       1560          LD     L,A
3A7C76   1570          LD     A,(CURCOL)
47       1580          LD     B,A
3E00     1590          LD     A,0
37       1600          SCF
04       1610          INC    B
1F       1620 PIXPL1   RRA
10FD     1630          DJNZ   PIXPL1
C9       1640          RET
         1650
3A7A76   1660 DRCHAR   LD     A,(CURCHR)
6F       1670          LD     L,A
```

```
2610      1680         LD     H,16
29        1690         ADD    HL,HL
29        1700         ADD    HL,HL
29        1710         ADD    HL,HL
EB        1720         EX     DE,HL
215AC1    1730         LD     HL,49498
0608      1740         LD     B,8
1A        1750 DRCHL1  LD     A,(DE)
E6F0      1760         AND    240
77        1770         LD     (HL),A
23        1780         INC    HL
1A        1790         LD     A,(DE)
17        1800         RLA
17        1810         RLA
17        1820         RLA
17        1830         RLA
E6F0      1840         AND    240
77        1850         LD     (HL),A
13        1860         INC    DE
2B        1870         DEC    HL
7C        1880         LD     A,H
C608      1890         ADD    A,8
67        1900         LD     H,A
10EB      1910         DJNZ   DRCHL1
C9        1920         RET
00        1930 CURCHR  DEFB   0
00        1940 CURLIN  DEFB   0
00        1950 CURCOL  DEFB   0
F0F0C030  1960 CHAR1   DEFB   240,240,192,48
C030C030  1970         DEFB   192,48,192,48
C030C030  1980         DEFB   192,48,192,48
C030F0F0  1990         DEFB   192,48,240,240
F0F0F0F0  2000 CHAR2   DEFB   240,240,240,240
F0F0F0F0  2010         DEFB   240,240,240,240
F0F0F0F0  2020         DEFB   240,240,240,240
F0F0F0F0  2030         DEFB   240,240,240,240
```

Après avoir entré les codes, on tape la touche Q pour sortir du programme ENTHEX. Pour sauvegarder le programme, on tape :

SAVE"CHDES,B,30000,365

Pour le recharger, il suffit de taper :

LOAD"",30000

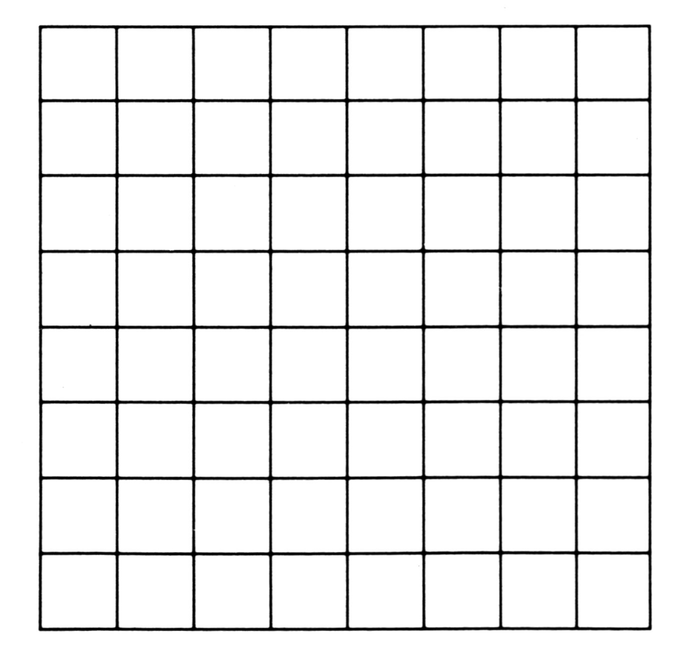
Ce programme est appelé par l'instruction CALL 30000. Une grille de 8 sur 8 apparaît à l'écran avec le nombre 0. Les caractères dont le code est compris entre 0 et 31 sont des caractères spéciaux et ne peuvent être redessinés. En tapant la touche N, on passe au caractère suivant, alors que la touche P permet de revenir au caractère précédent. Un curseur est affiché à l'intérieur de la grille ; on le déplace à l'aide des touches de déplacement du curseur. En tapant la barre d'espacement, on affiche ou on efface un pixel à l'intérieur d'une case. Lorsqu'un caractère a été dessiné, on peut le sauvegarder en tapant la touche ENTER pour sortir du programme, puis la commande :

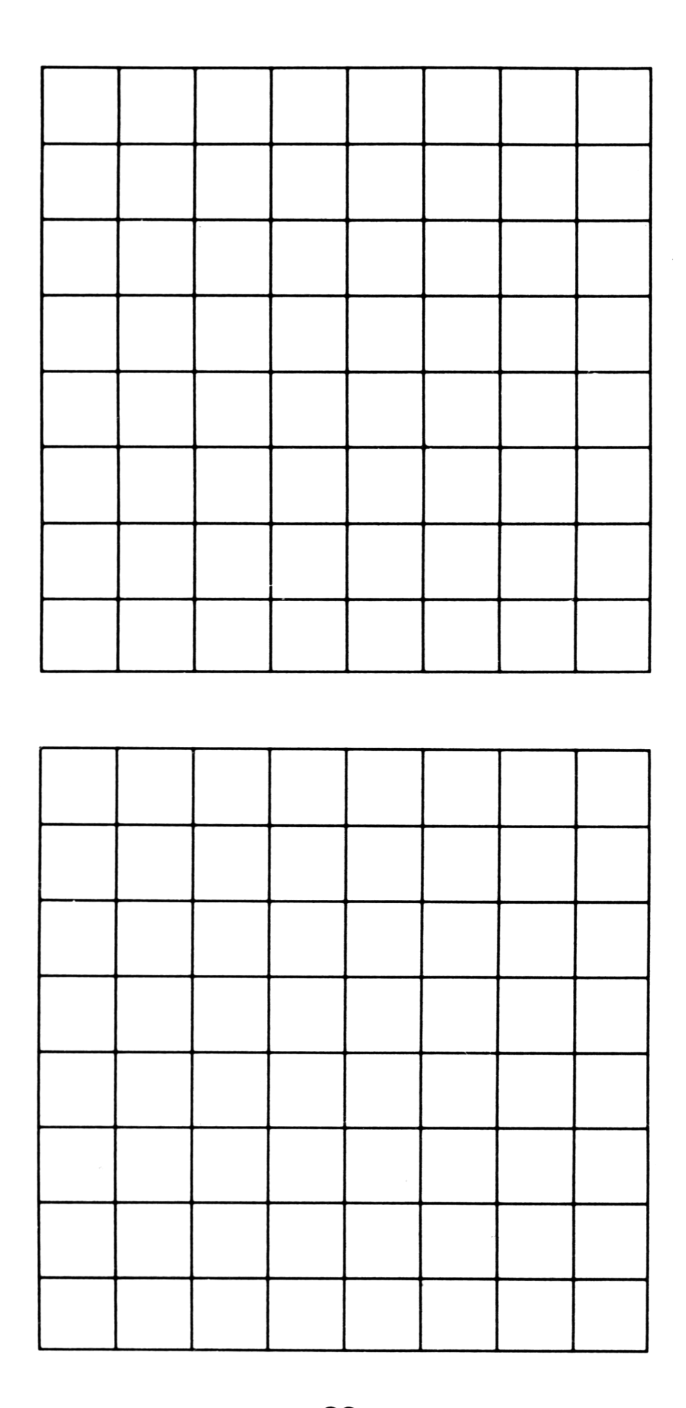
SAVE"CHARS",B,32768,2048

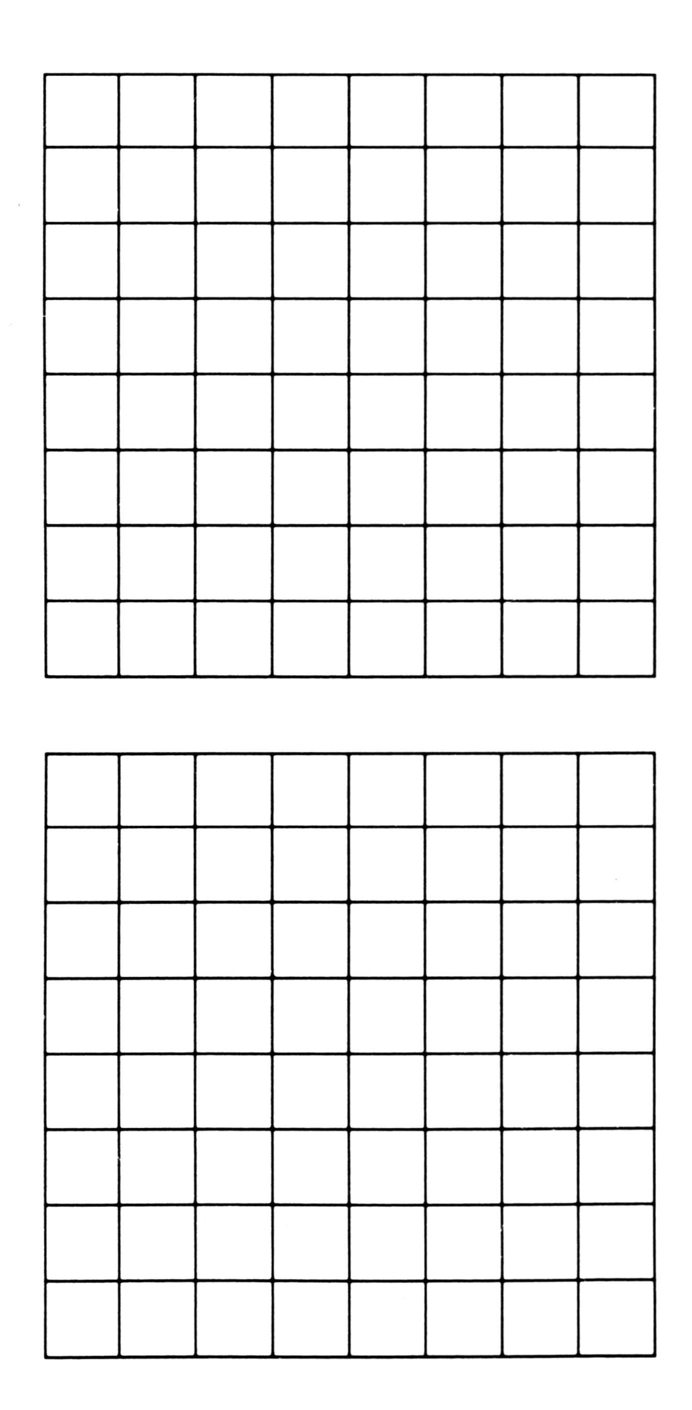
Pour recharger un caractère, on tape :

X=HIMEM +1 <**ENTER**>
LOAD"CHARS",X

Ces grilles vous aideront à dessiner vos caractères.







RÉPONSES AUX QUESTIONS

1. La partie de poids fort du nombre décimal 45621 est 178, sa partie de poids faible étant 53.

2. Le nombre décimal ayant 64 pour partie de poids fort et 31 pour partie de poids faible, est 16415.

3. Si les octets d'adresses 40000 et 40001 contiennent respectivement les valeurs 5d et 15d, le registre HL contiendra la valeur 3845 après exécution de l'instruction LD HL, (40000).

4. Si le registre HL contient la valeur 35621 et que l'instruction LD (40000),HL est exécutée, la valeur 37 est alors chargée à l'adresse 40000 et la valeur 139 à l'adresse 40001.

5. L'équivalent décimal de E3h est 227d.

6. Si FBh est la partie de poids fort d'un nombre et CBh sa partie de poids faible, ce nombre vaut 64459d.

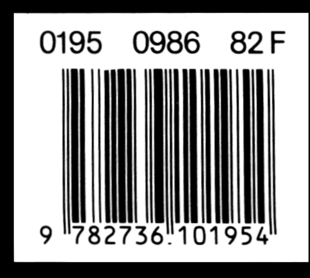
7. Si les bits 1, 4 et 6 d'un octet sont positionnés, celui-ci vaut 82 d.

8. Un octet correspondant à 67 a ses bits 0, 1 et 6 positionnés.



Achevé d'imprimer par l'imprimerie PRAXIE
Dépôt légal 3e trimestre 1986 - Imprimeur n° 1000

Les livres sur le BASIC sont nombreux, et le possesseur d'un micro-ordinateur est à même d'y trouver toutes les informations nécessaires pour débuter en programmation. Cependant, la réalisation de programmes demandant une grande vitesse d'exécution l'amène à se heurter rapidement à un problème majeur : la lenteur de l'interpréteur BASIC. Ce livre permet d'aller plus loin en abordant la programmation en langage machine sur l'Amstrad CPC 464. La façon de programmer l'équivalent des instructions BASIC PRINT, GOTO, GOSUB, FOR/NEXT, etc. est tout d'abord étudiée, puis ces notions sont appliquées à la réalisation d'un jeu d'action. De nombreux sous-programmes pourront être réutilisés par le lecteur dans ses propres programmes.



SYBEX

AMSTRAD PROGRAMMES EN LANGAGE MACHINE



Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/